1990-06

# Issues in expanding the Software Base Management System Supporting the Computer Aided Prototyping System

Huskins, James M.

Monterey, California. Naval Postgraduate School

# NAVAL POSTGRADUATE SCHOOL
## Monterey, California


# THESIS

**ISSUES IN EXPANDING THE SOFTWARE BASE MANAGEMENT SYSTEM SUPPORTING THE COMPUTER AIDED PROTOTYPING SYSTEM**

by

James M. Huskins

Thesis Advisor: June 1990 Luqi

Approved for public release; distribution is unlimited.

# REPORT DOCUMENTATION PAGE

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S) |
| 6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): 52 | 7a. NAME OF MONITORING ORGANIZATION: National Science Foundation |
| 6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943 | 7b. ADDRESS (City, State, and ZIP Code): Washington, D.C. 20550 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION: Naval Postgraduate School; 8b. OFFICE SYMBOL (if applicable): | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: NSF CCR-8710737 |
| 8c. ADDRESS (City, State, and ZIP Code): Monterey CA 93943 | 10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO. |

11. TITLE (Include Security Classification)
ISSUES IN EXPANDING THE SOFTWARE BASE MANAGEMENT SYSTEM SUPPORTING THE COMPUTER AIDED PROTOTYPING SYSTEM.

12. PERSONAL AUTHOR(S)
JAMES M. HUSKINS

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Master's Thesis | FROM 10/89 TO 06/90 | June 1990 | 116 |

16. SUPPLEMENTARY NOTATION The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

| 17. COSATI CODES | | |
|---|---|---|
| FIELD | GROUP | SUB-GROUP |
| | | |
| | | |

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
rapid prototyping, models, programming languages, domain analysis, object-oriented database, software reuse

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This thesis proposes reorganizing the Software Base Management System of the Computer Aided Prototyping System (CAPS) to take better advantage of object-oriented database technology, domain analysis and rule based systems. A method for using the Prototyping System Description Language (PSDL) augmented with domain dependent keywords to classify reusable Ada components and organize them in an object-oriented database is presented. A rule based structure needed to implement this software base is also described. Implementation of this structure is a goal for further research.

| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: [X] UNCLASSIFIED/UNLIMITED [ ] SAME AS RPT. [ ] DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL: Luqi | 22b. TELEPHONE (Include Area Code): (408) 646-2735; 22c. OFFICE SYMBOL: 52Lq |

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

Approved for public release; distribution is unlimited

# ISSUES IN EXPANDING THE SOFTWARE BASE MANAGEMENT SYSTEM SUPPORTING THE COMPUTER AIDED PROTOTYPING SYSTEM.

by
James M.Huskins
Major, United States Army
B.S., United States Military Academy, 1977
M.S.E., Catholic University of America, 1987

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

**NAVAL POSTGRADUATE SCHOOL**
June 1990

Author: 
James M. Huskins

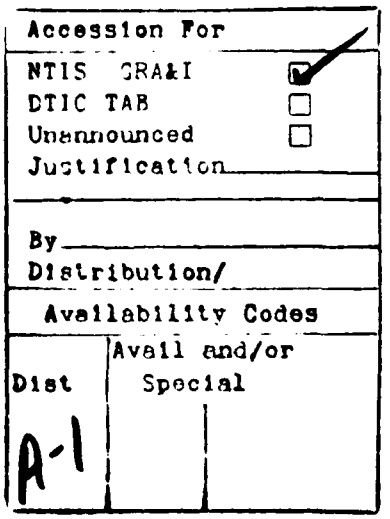
Approved By: 
Luqi, Thesis Advisor


Valdis Berzins, Second Reader


Robert B. McGhee, Chairman,
Department of Computer Science

## ABSTRACT

This thesis proposes reorganizing the Software Base Management System of the Computer Aided Prototyping System (CAPS) to take better advantage of object-oriented database technology, domain analysis and rule based systems. A method for using the Prototyping System Description Language (PSDL) augmented with domain dependent keywords to classify reusable Ada components and organize them in an object-oriented database is presented. A rule based structure needed to implement this software base is also described. Implementation of this structure is a goal for further research.

## THESIS DISCLAIMER

Ada is a trademark of the U.S. government Ada Joint Program office. Ontos is a commercial product of Ontologic Inc., Burlington, Massachusetts.

## TABLE OF CONTENTS

# I. INTRODUCTION

This thesis addresses issues related to the reuse of software components in support of a computer aided rapid prototyping environment. The specific system addressed in this thesis is the Software Base component of the Software Database System, a subsystem of the Computer Aided Prototyping System (CAPS). This portion of the Software Database System stores previously developed Ada components for potential reuse by other programs or subprograms.

This chapter provides background on some of the issues related to reuse of software in general and the applicability of reuse to the rapid prototyping environment provided by CAPS. Succeeding chapters present an overview of reuse concepts including some of the more promising methods and research in this area. This is followed by an evaluation of the current state of the Software Base component of the CAPS system and the use of the Prototyping System Description Language (PSDL) as the basis for classification and retrieval of software components from the reusable software base. Domain analysis and its applicability to software development will be discussed via an example, the Common Ada Missile Parts (CAMP) project sponsored by the Air Force and performed by McDonnell Douglas Corporation from 1984-1988 [Ref. 1]. A conceptual design of a software base incorporating the results of the CAMP domain analysis, object-

oriented concepts, and use of an augmented form of PSDL as the base language for classification and retrieval of components is presented and contrasted with the CAMP methodology.

This thesis addresses the specific problem of integrating reuse of Ada components into CAPS. However, the methodology for development of the object hierarchy, the basic rules for search path derivation, and the pattern matching process used in the transformation of the augmented PSDL specification to a set of candidate reuse classes, could easily be adapted to reusable components in other programming languages or other reusable entities. Some suggestions of other reusable entities which may become candidates for inclusion in the Software Base are included in the concluding section of this thesis.

## A. THE SOFTWARE CRISIS

The demand for larger and more complex software systems continues to increase. Software development is a complex process. Programming large software systems involves teams of developers, working on various parts of the system. Coordination of this effort is an immense task. Our inability to deal with this complexity often results in software projects that are late, over budget, and deficient in their stated requirements. The problem involves not only the high cost of software development, but also in the poor quality of existing software [Ref. 2: p. 3]. Modification is so difficult that maintenance

soaks up more than half of the total resources. This state of affairs is sometimes called the "software crisis" [Ref. 3: p. 243].

While all acknowledge that the problem is there, few have any short term solutions. In fact, it is estimated by many that we are still 10-15 years away from having the tools we need to build the systems we would like to have now in a reliable manner [Ref. 4]. Research on the solutions to the software crisis seems to concentrate on three main areas: programming and specification languages, prototyping and knowledge based development, and reuse of software artifacts. Most development work on advanced software environments includes elements of all of these areas but few are concentrating on the integration of all of these areas into a comprehensive environment which will have any short term (within the next five to ten years) impact.

## B. RELATED AREAS OF RESEARCH

The next sections briefly describe some of the major areas of research involved in software development. All have potential long range benefits, but many are in the early stages of development.

### 1. Programming and Specification Languages

This area addresses the support of sound software engineering practices through the development of languages that support the development of reliable systems. This includes both High Order Languages (HOL) that support the conventional development and Very High Level Languages (VHLL) used primarily in artificial

intelligence. In the area of HOL support the Department of Defense took the lead with the decision to designate Ada as the language for DoD systems. The development of Ada as a language that offers great expressive power, is tightly controlled as a language, and demands the use of sound software engineering practices is already showing benefits in the development of software within DoD [Ref. 2: p. 8]. Very High Level Languages have great expressive power and most allow the use of symbolic expressions to represent functions and rules that support the development of expert systems. They have proven to have the most benefit in limited, domain specific areas where knowledge can be represented in the form of facts and rules. They have been used to a limited extent in the software development and show great promise, particularly in the area of specification assistance. The most notable efforts in this area are the Programmers Apprentice project at MIT and the Specification Assistant portion of the Knowledge Based Software Assistant project at Rome Air Development Center [Ref. 4]. Other notable VHLL's include REFINE developed and distributed by Reasoning Systems Inc. and GIST developed by the ISI at the University of Southern California [Ref. 4].

### 2. Prototyping and the Use of Expert Systems

Most of the major efforts in this area are focused on very narrow domains. Prototyping is used in two primary contexts: as a tool to define user interfaces, or as a method to test all or part of a system prior to entering a full scale development effort. A great deal of the effort involved in prototyping is on the validation of user requirements

at an early stage of system development. This validation has the potential to greatly reduce both the cost and time required to develop systems. In the context of software development expert systems are used to aid the process through the use of knowledge about the process as well as knowledge about specific problem domains. It is anticipated that many of the database intensive activities being performed today will be replaced by expert systems and knowledge based techniques in the future.

### 3. Software Reuse

Software reusability is widely believed to be a key to improving software development productivity and quality [Ref. 4]. It also has the most potential in the short term to show tangible benefits in a number of domains of application. It allows the developer to write fewer lines of code and to spend less time organizing. To date the promise offered by reusability is largely unfulfilled. Three factors that now make it practical to formalize a model of reusable software [Ref. 2] :

1. Maturation of the software industry has resulted in an accepted body of knowledge about data structures and algorithms.

2. A number of software engineering principles that help us deal with the problem of developing massive software intensive systems have been recognized.

3. The development of Ada a language that offers great expressive power, is tightly controlled as a language, and demands the use of sound software engineering practices.

Research in this area centers on two types of reusability technologies: composition technologies and generation technologies. In composition technologies the components

to be reused are largely atomic, and ideally unchanged during the course of their reuse. They can be thought of as building blocks that are combined to form larger programs (the process of composition). Generation technologies use reusable pattern information and weave the patterns together to generate new code [Ref. 4]. Other problems of reuse receiving a great deal of attention are the classification of components and organization of libraries and software databases so that needed components can be found when needed. These latter two problems which are the focus of later sections of this thesis.

## C. THE COMPUTER AIDED PROTOTYPING SYSTEM

The computer aided prototyping system is an integrated environment aimed at rapidly prototyping hard real-time embedded systems [Ref. 5]. This system differs in its approach from most other prototyping systems in its approach to software generation from a combination of translation and reuse methods. In its current state its approach to reuse most closely resembles the composition approach described in the previous section. However, the system also has a translation subsystem which is used to generate modules from the PSDL specification. Therefore there is not a need for extremely sophisticated composition systems which rely on low level components being combined into higher level program units. CAPS instead relies on the software base component to reduce the amount of work required of the prototyper and the translator by storing reusable components that are known to be efficient, reliable pieces of code that can be described using the prototyping language. These components may be at any level of detail, and

represent entire subsystems consisting of many lines of code. The challenge is to organize the database of reusable component objects in a manner that the database can be queried during a user session, using only the information available in the PSDL description, and rapidly return a matching component or a message to the user that the component is not available.

CAPS has a great deal of potential for short term benefit. Its architecture is also open enough that it can take advantage of the results of emerging technologies in many of its subsystems with minimum disturbance to the other subsystems. The prototyping language (PSDL) is a relatively small, yet powerful, language which should allow the system to represent constructs in many domains. Domains can be described by augmenting PSDL with domain specific keywords. This feature of PSDL has not been tested in any previous research. This thesis presents a method of using an augmented form of PSDL for both classification and retrieval of reusable software components.

## D. GOALS OF THIS THESIS

The goal of this thesis is to extend the current conceptual design of the reusable software component base to more adequately meet the user's needs in the CAPS prototyping environment. The structure of the objects used to encapsulate the reusable components is detailed, including the attributes and methods which may be needed as the CAPS system evolves. Domain analysis is incorporated into the system through the augmentation of the PSDL language with keywords. A methodology for adding new

classes to the database with minimum disturbance of the existing hierarchy or data objects is also developed in this thesis. Finally, a new, more generalized, method for searching the database based on some simple rules stored in a specialized object class is presented. The final design of the software base presented in this thesis is based on integrating features of domain analysis, object-oriented databases, knowledge based systems, and language specific aspects of a specific programming language, in this case Ada, to support a prototyping environment using a higher order language, in this case PSDL. The focus is on the processing required to translate a PSDL specification into a database query in a form that reduces the search to the minimum required in support of the functional environment identified by the prototyper.

# II. BACKGROUND

This chapter contains material on the development of an effective mechanism for reuse of components in a rapid prototyping environment. A brief discussion of the necessary and sufficient conditions for reuse is followed by an overview of some of the more prominent strategies used in organization of libraries and databases of reusable components. A more detailed discussion of reuse issues can be found in previous thesis work by Galik [Ref. 6] and Steigerwald [Ref. 7],the CAMP overview document [Ref. 1], and various other articles [Refs. 2,3]. The major types of software generation environments and their approaches to reuse as well as integration mechanisms are discussed in some detail. Since the major area of concern of this thesis is the problem of improving and more effectively integrating reuse into the CAPS system using object-oriented database technology, an overview of object-oriented concepts and databases, the current status of the database of reusable components supporting CAPS, and its position in the overall CAPS environment are presented here as well. In subsequent chapters the Prototyping System Description Language and its relation to the Ada language is discussed. PSDL is also evaluated for its potential as a classification and retrieval language for reusable Ada components. The concepts presented here will be used in this discussion.

## A. CONDITIONS NECESSARY FOR EFFECTIVE REUSE

In order to effectively reuse software components in any environment there are several conditions that must be met. These operational problems fall into four main areas and include [Ref. 4: p. 5]:

1) Finding components
2) Understanding components
3) Modifying components
4) Composing components

In this thesis the focus of effort is on finding and understanding components. Finding components is more than just finding an exact match. It includes locating highly similar components. It includes the classification of components included in the software base and the organization of these components in a order to aid in search and retrieval operations. Understanding components is primarily a documentation issue. But the representation of descriptive information in a manner that can be useful to an interactive user is of concern here. Modification and composition of components is related to how components become parts of systems. This area is briefly addressed in this thesis. CAPS can support these operations. Since CAPS is a prototyping environment rather than a production environment or Software Generation System, the actual production of code is not the primary objective of the system. As CAPS continues to evolve these areas will

become more important. The most important objective of the Software Base Management System is the identification of potentially reusable code. Integration of the reusable components into larger systems and programs is handled by other parts of the CAPS system.

## B. LIBRARY CLASSIFICATION SCHEMES

Reuse mechanisms can be found as a component of a number of software development environments. The most basic means of incorporating reuse into a software generation system is through the use of libraries. In a library of reusable components a group of software components that support particular programming functions or narrow domains of interest are placed in some common areas that are easily accessible by the programmer. The most common example of libraries are the common functions found in C libraries, and narrowly defined domains, such as math libraries in C and Ada programming environments. These libraries are very useful, but have a major drawback. These simple mechanisms are not automatically accessible without foreknowledge of their composition. The design of an effective reuse system must allow the programmer to identify the potentially reusable components without actually knowing of their existence. This is the main problem addressed in the design of classification and retrieval mechanisms.

Two schemes of classification are continually mentioned in discussions of organization of reusable components. The most commonly mentioned is the Booch

Taxonomy of Ada components [Ref. 2]. Booch uses a descriptive method based on the form of the component including information on the abstraction of the component and its time and space properties [Ref. 2: p. 36]. His method is very useful in the general categorization of components into their basic data structure related types. The other method of classification developed by Prieto-Diaz takes a different approach. The Prieto-Diaz scheme is based on organizing components to be processed using the relational database model. He describes a tuple of six attributes that can be used to categorize any component. Three of these are related to the functionality of the component and three related to the environment [Ref. 8: pp. 280-281].

## C. INTEGRATION OF REUSE INTO ENVIRONMENTS

Classification and organization of components are only two issues related to the use of reusable components in programming environments. There must also be some means of integrating the components into the target program either in their stored form or in some modified form. These are the problems of understandability and modification addressed earlier. In this section a description of the approaches to integration of components into various types of environments is presented.

### 1. Types of Reuse in Existing Environments

The major types of reuse found in existing programming environments are based the approaches used in these systems. The major types of software generation systems in use today are based on either the generation of components from some high

level language or the composition of existing components into larger programs and systems. Systems using the generation approach attempt to generate source code through the transformation of some higher order language into source level programming language constructs using knowledge based techniques to control the transformation. The reuse of transformation information stored in a database of programming knowledge is used to accomplish the code generation. Systems developed using this approach have thus far been inefficient and inadequate to the needs of embedded and hard real-time programs. But there is a great deal of ongoing research in this area and it is generally acknowledged that generation based systems are the systems of the future [Ref. 3].

Systems that have the most short term potential benefit are composition based. Systems developed using the composition approach attempt to identify source components and combine them into larger programs. Identification and integration can be guided by programs that access the database or by knowledge based techniques. These systems can store multiple versions of the same component and can generate much more efficient systems. However the problem is how to decide the level of abstraction for the individual components. Storage of large subsystems can provide gains in productivity at the cost of overall program efficiency. Using large subsystems can result in the incorporation of unused code into a program. Using small components is acceptable for small programs, but the problem of how to find components in a large database of very small components and combine them is much more complex. The programmer needs to understand more

about the details of the software base in order to adequately specify needed components. The database designer has the problem of designing efficient methods for search, selection and combination of the components. Another drawback of composition based systems in general is the lack of any means of building components from scratch. If the component is not in the library or database it provides only a message to the user.

### 2. Domain Analysis

Another approach to the organization and integration of reusable components into a software generation system is to organize the reuse effort around the projected application area or domain of interest. Using a process known as domain analysis [Refs. 1,8,9] the application area is broken down into the categories of subprograms or program objects needed to implement needed functions and structures for parts of a system. A domain analysis involves an intensive study of the application area and previous software development efforts [Ref. 1: p. 7]. Products of the domain analysis may include parts taxonomies or classification schemes for storing reusable components in databases, domain languages for specifying software components, transformation rules used for generating components from the specifications, and rules for composing subsystems from combinations of smaller software components. This is the approach used in Draco and the CAMP project. A more detailed summary of the process used by these two methods and the products of domain analysis in each of these systems is described in the following sections.

### *a. The Draco methodology*

The Draco method involves the analysis of an application area, or domain, in order to identify and codify the reusable concepts or objects, and models into a domain language that can be interpreted by the Draco system and used in the generation of code. The Draco method uses three analysts: an application domain analyst, a domain designer, and a modeling domain analyst. The application domain analyst examines the needs and requirements of systems in come common application domain, identifying the objects and operations that are germane to an area of interest. Once identified this is given to the domain designer who specifies the objects and operations in a manner known to the Draco system. The modeling domain analyst is concerned with the methods used to model domains in Draco. The primary concern is to insure that the system does not duplicate existing domain descriptions when describing new domains.[Ref. 4: p. 302]

The end result of the analysis is a domain description of a particular domain in a language that can interpreted by the Draco system. The Draco system is a tool that combines generation and composition approaches by successively transforming statements of the component specification in the domain language. Statements in the domain language are parsed into internal form and may be [Ref. 4: pp. 306-307]:

1. prettyprinted back into the external syntax of the domain;
2. optimized into a statement in the same domain language;

3. taken as an input to a program generator that restates the problem in the same domain;

4. analyzed for possible leads for optimization, generation,or refinement; or

5. implemented by software components, each of which contains multiple refinements and which make implementation decisions by restating problem in other domain languages.

The Draco system recognizes the use of actual source code as valuable to short term productivity but is oriented toward generation technologies as the long term solution [Ref. 4: p. 316]. However they do recognize the need for analysis at the major subsystem level as a means of constraining and smoothing the modelling process, particularly in the early stages of design. The reuse of design and modelling information that may still be modified prior to the production of actual code is limited to subsystems that are already known, or previously developed and stored in Draco [Ref. 4: pp. 315-316].

### *b. The CAMP methodology.*

One of the objectives of the CAMP project was to evaluate the utility of source code reuse to the maximum extent. Their domain analysis was oriented toward the identification of the areas of missile software systems that had commonality. They did this by studying ten previously developed missile related software systems. The result of this domain analysis and commonality study was the identification of seven major areas in the domain of interest. Within these areas a total of 21 categories of components were developed. The CAMP parts taxonomy is based on classifying components into

these 21 categories. The taxonomy is shown in Figure 2.1 [Ref. 1]. The category of each part is an attribute in the database of CAMP parts which was developed using Oracle, another package using the relational model.

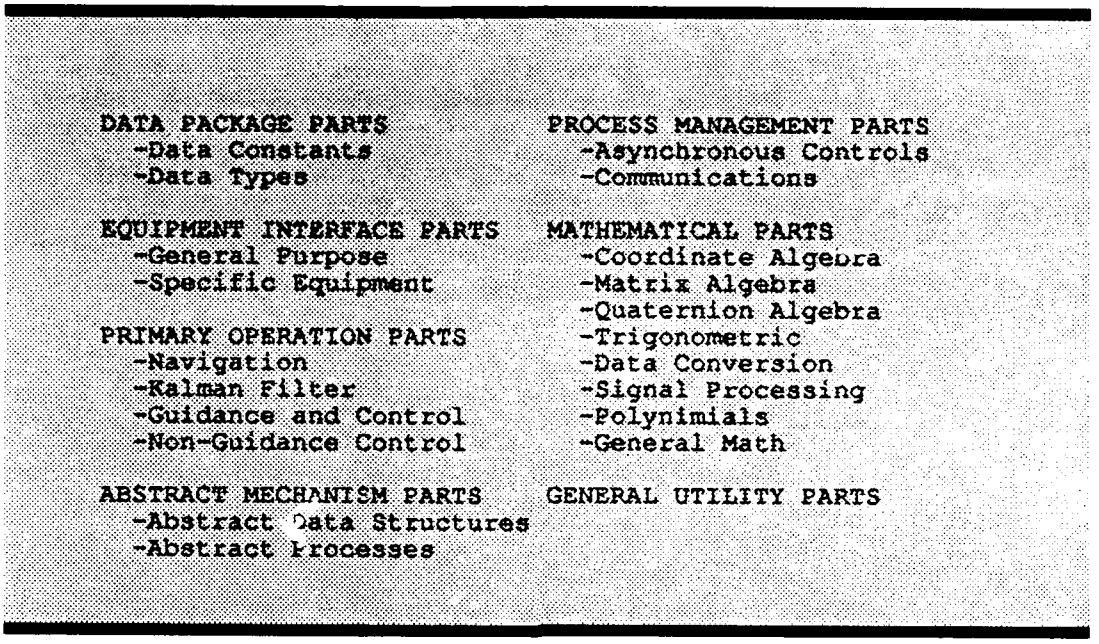



**Figure 2.1 The CAMP Parts Taxonomy.**

Parts that were developed under the CAMP project were largely source code using the Ada generic package capability, some specific data types and operations common to the majority of the systems studied, and schematic parts, or templates that can be called up and customized by user. The CAMP approach is oriented toward composing systems by retrieving parts from a software base and combining them into systems or subsystems. The composition uses an expert system containing database search rules and combination

rules for the individual components and rules for combining components into subprograms. More detail on CAMP is presented in Appendix A.

## D. AN IDEAL SOFTWARE GENERATION SYSTEM

In previous sections the two general categories of software generation systems were described. There are some obvious tradeoffs made when building a system using either approach. An ideal system would include both of these approaches allowing for the combination of the benefits of each. There would be components of this system that would be able to combine reusable source level components as well as generation components used to build new components by transforming specifications. The structure of such a system was described by the CAMP project developers and is shown in Figure 2.2 [Ref. 1: p. 50].

The ideal system would be controlled by an expert system containing both application domain and programming knowledge. It would have database and knowledge base components. And it would have the capability to generate software from an input requirement through either, or a combination of generation and composition techniques.

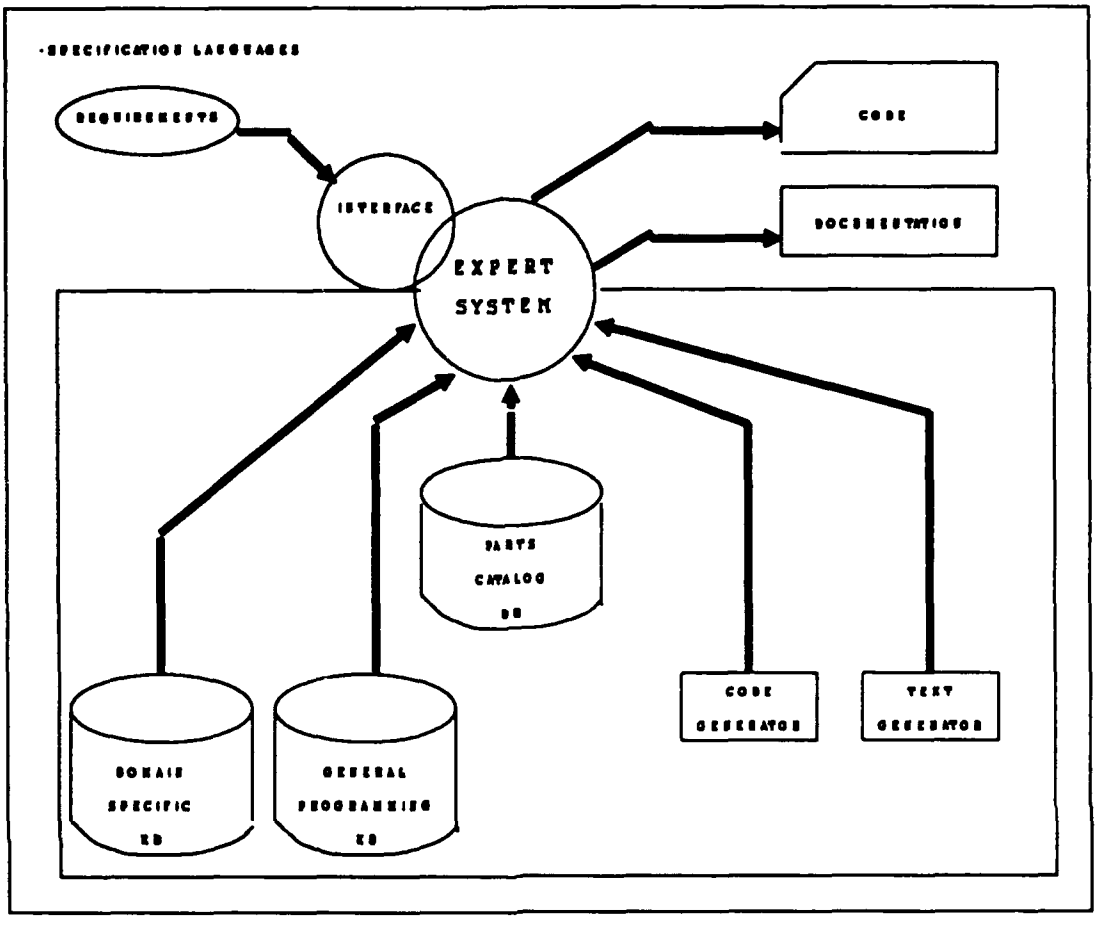



**Figure 2.2 An Ideal Software Generation System**

## E. THE COMPUTER AIDED PROTOTYPING SYSTEM ENVIRONMENT

The Computer Aided Prototyping System (CAPS) is a tool which is being developed to support software development through the implementation of a rapid prototyping methodology [Refs. 10,11]. CAPS is designed to aid the software designer in the analysis of hard, real-time systems using specifications and reusable software components to automate the rapid prototyping process. It uses a high level prototyping language called

the Prototyping System Descriptive Language (PSDL) [Ref. 12]. Its major goals are to decrease the development time and increase the quality of production code by iteratively prototyping the system and/or key subsystems until the requirements and specifications needed to build the production system are firm. The major components of the system are shown in Figure 2.3 [Ref. 6: p.9].

As previously mentioned, software generation is not the primary goal of CAPS. However the identification of reusable software components can greatly aid in both the reduction of development time and accuracy goals of CAPS. CAPS reusable components are stored as objects representing the encapsulation of various programming structures

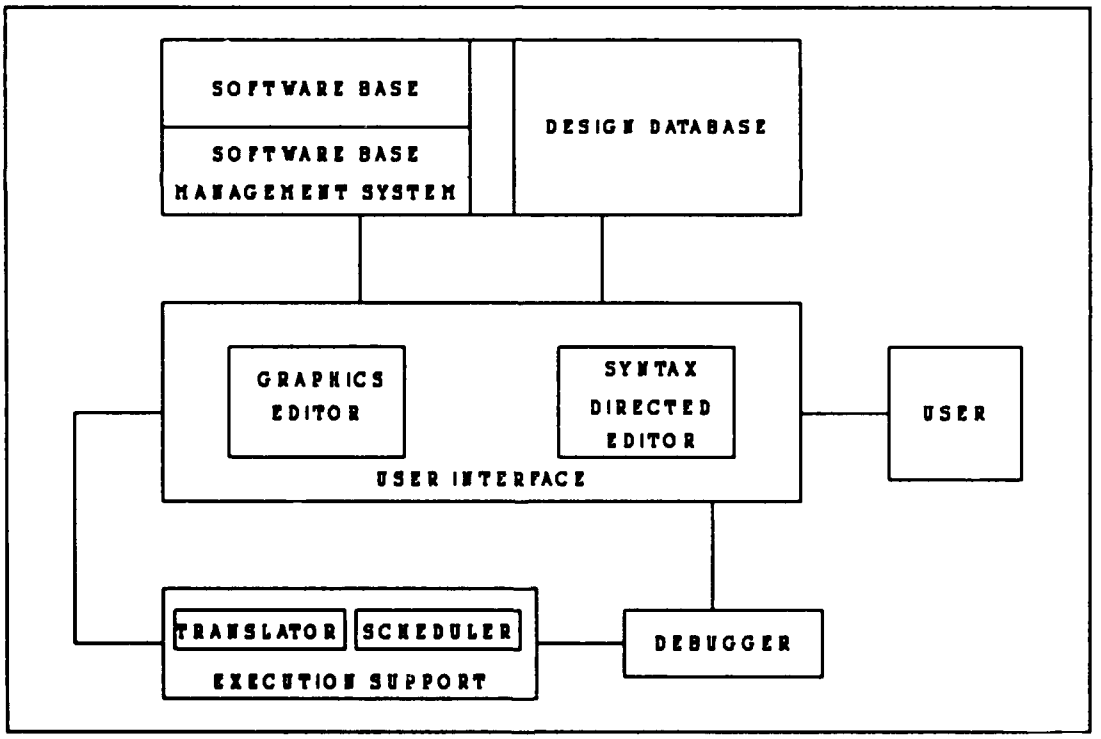



**Figure 2.3 The Computer Aided Prototyping System**

which are of use to the designer in choosing the data types and operations needed to implement programs in the given application domain. The software base component is currently implemented as a collection of Ada related objects stored in an object-oriented database. Actual Ada source code is an attribute of the stored object as well as other information needed for matching PSDL descriptions to corresponding Ada implementations. For any given domain the number of objects which may be stored is large. The current database organization is shown in Figure 2.4 [Ref. 6]. It provides for only two classes, or groupings of objects. This makes the problem of finding objects a very long and difficult one. This thesis concentrates on ways to improve the organization of the database to support identification and retrieval of reusable objects.

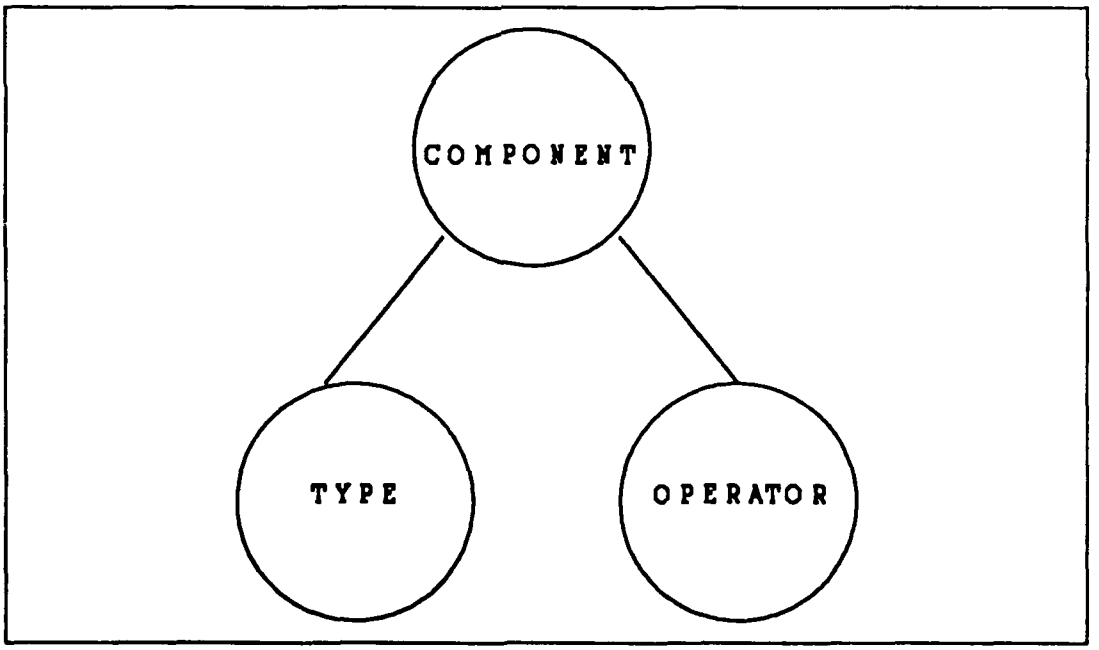



**Figure 2.4 The Current Software Base Organization**

## F. OBJECT-ORIENTED DATABASES

Object-oriented databases have become increasingly popular as an element of design environments. The object oriented paradigm has shown particular promise in the area of computer assisted design and manufacturing environments (CAD/CAM) [Ref. 13]. Object-oriented databases are also becoming more common as components of software development environments. Therefore it is appropriate to discuss this area in some detail. The following discussion of object-oriented concepts further justifies the continued inclusion of this type of database for the Software Base Management System of CAPS.

### 1. Object-Oriented Concepts

Before explaining the object-oriented database in more detail a brief review of object-oriented concepts is required. One accepted definition of object-oriented is [Ref. 14]:

**object-oriented = objects + classes + inheritance**

The concept of an object is based on the principles of data abstraction and information hiding [Ref 15: p. 2]. Data abstraction represents a system as a set of objects and the set of operations characterizing the behavior of the objects [Ref. 15: p. 2]. Information hiding decomposes a system into components, each characterized by knowledge of a design decision hidden from all others [Ref. 15: p. 3]. Classes are templates from which individual instances of objects may be created by "create" or "new" operations. Classes are organized in hierarchies to facilitate the inheritance of attributes and methods used by

that class. Inheritance is a mechanism which allows a class to inherit operations from superclasses and also allows subclasses of the class to inherit its operations [Ref. 14]. Theoretically a given class of objects may have many subclasses or superclasses. However, practically, most languages implementing object-oriented concepts organize the class structure into a hierarchy using only single inheritance. An object of a given class may have many subclasses, but only one directly connected superclass.

Object-oriented approaches were first used as a means of exploiting encapsulation. Encapsulation is necessary to ease the development and maintenance of larger systems by decomposing the larger system into smaller encapsulated subsystems. Typically, the encapsulated subsystems are thought of in terms of "objects" rather than "programs" and "data" [Ref 15: pp. 3-4]. You adopt a particular object model and then encapsulate objects in terms of a visible interface, called operations, while hiding the object's implementations and data structures. Just how object-oriented languages provide constructs for defining useful kinds of objects can be discovered in an analysis of the issues of software reusability. Reusability of programming objects is supported by the packaging of objects in such a way that they can be conveniently reused without modification to solve new problems [Ref. 15: p. 5].

Object-oriented development is the use of object-oriented concepts in the analysis, design, and implementation of software systems [Ref. 15: p. 3]. Object-oriented development leads to architectures that are based on systems manipulating objects. Rather

than concentrate on what the system does, using this method of development the concentration is on what the parts of the system do or have done to them by other parts [Ref. 14].

### 2. The Class Hierarchy of Objects

In object-oriented programming one of the main advantages is the ability to reuse existing attributes and methods defined for a given class of objects in the subclasses of that object. Often the definition of new object classes is accomplished by refinement or specialization of already existing classes. The newly defined object class inherits all or many of the characteristics (attributes and methods) of its superclass and may either add or refine methods in its own definition. A sample inheritance hierarchy is shown in Figure 2.5 [Ref. 15].

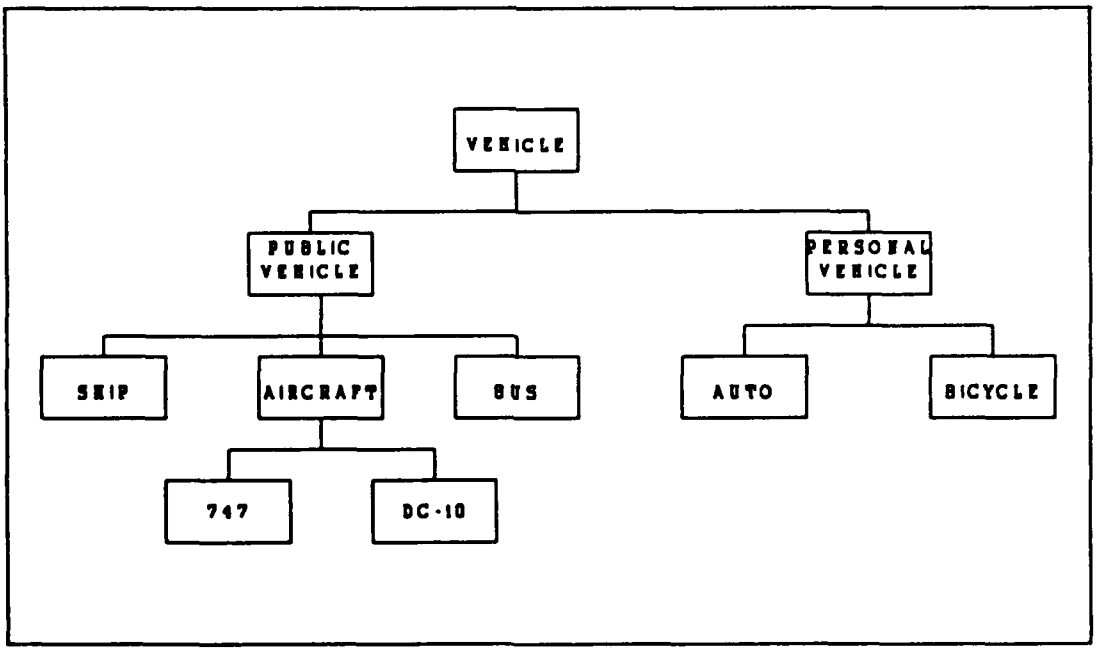



**Figure 2.5 A sample Class Hierarchy of Objects**

In this example the entire hierarchy is based on the specialization of the objects by adding more detail as we go down the hierarchy. The objects at the lower levels inherit all of the characteristics of the higher levels. But the characteristics of the more general superclasses remain available to, or are inherited by, the more specialized object classes. This is the idea of the "is-a" relationship used as the basis for classification in object-oriented systems. The Ship is-a public vehicle, that is, it is a specialization of the public vehicle class. It has all the characteristics of the public vehicle plus other specialized characteristics that distinguish it from its more general superclass.

### 3. Object-Oriented Programming and Rapid Prototyping

As previously described, object-oriented programming methodologies concentrate on the use of inheritance for code reuse between objects within the same parts of a hierarchy. They also focus on the use of encapsulation to insure that individual objects are as independent as possible. It is this second aspect that is the most important in the context we are discussing. In the general case, a program or program module can be described as an entity (object) that has some input and output, and performs some transformation. This description of a program module can form the basis for encapsulating ideas, or threads, that an individual program module represents. If this approach is taken throughout the design of a software system, the successive iterations of problem decomposition eventually form objects at a very low level, which capture

small subsets of the overall requirement into individual entities (objects). Thus an object may be represented by the function it performs.

### 4. A Database of Objects to Support Rapid Prototyping

With these basic definitions in mind, how do we merge the concepts of OOP languages with database management technology and rapid prototyping? Galik lists some of the more significant properties of an OODBMS [Ref. 6]. These include persistency, active data, extensibility, and abstraction. He points out that data objects must be "non-volatile" while maintained within the database [Ref. 6: p. 30]. A database requires that the persistence of its data "transcend" that of individual programs which may have a "short" lifetime. Since objects "persist" between execution of operations they should provide a better starting point for databases [Ref. 14]. The object-oriented database of components used in CAPS can be thought of as a persistent entity in its own right. The components that make up the database are only a portion of what we must consider when comparing this form of database with other database models. Additionally, we must consider the standard database functions which the OODBMS must provide. These functions include concurrency control, recovery, transaction management, and security found in all types of databases [Ref. 6: p. 30].

OODBMSs have many of the features necessary to aid in the retrieval and use of complex data applications that are particular to computer aided prototyping. The

following are some of the features identified in OODBMSs as aiding in rapid prototyping [Ref. 13: pp. 32-35]:

a. Increased modeling power over relational models. Relational models have trouble forcing real-world objects into fixed programming constructs and are often inadequate to store complex info.

b. A set of predefined system types (such as set, queue, stack, and list). Avoids cluttering the solution with data specific routines.

c. Stores the data structure (object) directly on the disk. Avoids extra code required for loading and mapping disk information to the usual memory data structures if the language supports object persistence.

d. Enforces data abstraction and data hiding. The programmer needs only to know what the objects do.

e. Supports code reusability. The programmer can write less code while implementing the same functionality.

f. "Triggers" modules to allow programmers to combine them as needed. It is hard to make code generic enough to achieve the same effect without objects.

g. Allows generic programming through polymorphism and meta-information. Polymorphism takes the parameters passed and automatically dispatches a call to the correct routine called. Meta-information alleviates the need to hard-code user-defined types in the program.

The following may be considered drawbacks of an OODBMS:

a. A long learning curve is associated with learning object programming.

b. Robust and reliable tools, such as source-level debuggers and fast compilers, are not readily available.

c. Without implementation level knowledge, a programmer may not recognize a bug hidden through abstraction.

d. Since semantic information is also stored with the object, it may require more storage space than regular file based systems.

e. Strict type checking causes compile time performance to be worse than in other database management systems.

## G. DATABASE SUPPORT OF CAPS

The CAPS environment is oriented toward the development of programs using an object-oriented approach that will be implemented using a conventional programming language, in this case Ada. Conventional programming languages have constructs that are used in implementing modules, that also have their own descriptions (functions and procedures, for example). This programming language information may also be useful in designing individual objects in the manner described above. But to effectively reuse objects designed using conventional programming languages there are other issues that must be addressed. One of the more important is how to store the objects for easy retrieval. This is the area of classification. One of the goals of building programs from encapsulated modules of code is to reuse code previously used in the program, or in similar programs developed in the past. This is where the use of object-oriented databases of reusable software components comes into the development process. The classification of reusable components into a hierarchy based on specialization of the component object classes and methods provides the needed indexing information as well as a code sharing (inheritance) capability for methods needed to use the objects.

Information presented in this chapter provides the needed guidelines for defining the structure of an object hierarchy of reusable components to support the CAPS system. Other parts of the CAPS system provide the aggregate constructor, or composition mechanism, needed to combine objects stored in the database into larger programs. Using the guidelines described in this chapter a proposed hierarchy of reusable components is developed and the needed attributes and methods for the objects are identified.

# III. PSDL AND THE CAPS ENVIRONMENT

In this chapter the Prototyping System Description Language (PSDL) is evaluated for its effectiveness and potential as a classification and retrieval language for the Software Base Management System. The Computer Aided Prototyping System is described in some detail to show the interactions between the system components, particularly those that interface with the Software Base Management System. A much more detailed description of this latter area is contained in [Ref. 5]. PSDL is extensively covered by [Refs. 12,16].

The focus of this chapter is on the development of an effective mechanism for organizing the software base using concepts supported by PSDL. We outline a classification scheme for storing components in the software base and the requirements for processing PSDL specifications into a form that can be used to query the database. This is followed by a discussion of a method of deriving a class hierarchy using PSDL constructs and the results of the domain analysis. Finally the problem of finding components in the class hierarchy is addressed through the development of some simple access rules used as a front end to the database. Pre-processing these rules narrows the search of the database.

## A. PSDL AS A CLASSIFICATION LANGUAGE

PSDL holds a great deal of promise as a classification and retrieval language for accessing the object oriented database of reusable components. It is a small, yet powerful, language and is very compatible in its method of specification with that of the Ada language. It can also be customized, through its keyword capability, to represent the domains of interest covered by the software base. Appendix B contains the current version of the basic PSDL grammar. In the following sections the use of PSDL specifications to represent component specifications is discussed with particular emphasis on the detail that is possible using basic PSDL augmented with a few keywords derived from a sample domain analysis.

### 1. Mapping PSDL Descriptions to Ada source code.

A PSDL description of a component involves two main parts: a SPECIFICATION and an IMPLEMENTATION. The SPECIFICATION part describes component attributes including the component type, either OPERATOR or TYPE, INPUT and OUTPUT variables, and information on requirements and constraints. The SPECIFICATION part contains much of the information seen in the specification portion of an Ada implementation of the part. A sample PSDL description with a corresponding implementation is shown in Figures 3.1 and 3.2. There are several areas of commonality between a PSDL specification and an Ada component used to implement the specification.

```
PSDL DESCRIPTION

OPERATOR controller_start_up
SPECIFICATION
    INPUT control_panel: real, sensor_input: real
    OUTPUT missile_control: real
    MAXIMUM EXECUTION TIME 90ms
    BY REQUIREMENTS control_panel_max
DESCRIPTION
{ Extracts the control panel input and sensor input and
  uses them to calculate the control signal for the missile.
}
END
```

**Figure 3.1 Sample PSDL Component Description.**

```
--ADA IMPLEMENTATION OF CONTROLLER_START_UP

WITH weapons_controller_package;
USE weapons_controller_package;
PROCEDURE controller_start_up(control_panel,
                              sensor_input: IN float;
                              missile_control: OUT float)is
  enable_signal: real;

BEGIN
   enable_signal:=calculate_control(control_panel,
                                    sensor_input);
   missile_control:= enable_signal;
END controller start_up
```

**Figure 3.2 Sample Implementation of Figure 3.1**

These areas can be useful in choosing reusable implementations meeting the requirements described in a PSDL description. The most obvious areas of commonality are the direct mapping of the types used in the specification and the implementation. The requirements information stored in the component specification is useful as it may be possible to identify other needed components from information contained in the requirements. In this case the reference to the control_panel_max requirement is handled by the weapons_controller_package. This package is referenced using the Ada WITH and USE capabilities in the implementation of the component. The MAXIMUM EXECUTION TIME specified may also be useful in choosing between possible implementations of this component having the same name and capabilities, but different timing properties. The information in the DESCRIPTION portion of the specification may be useful to the user of a browsing system for discriminating between components by matching the DESCRIPTION used in the PSDL component with documentation representing the functionality of actual components.

### 2. Using PSDL to classify components

PSDL has several constructs included in the language that are useful for grouping components. The areas that are important in the use of PSDL to classify objects are the ability to classify objects as either OPERATORs or TYPEs, the INPUT and OUTPUT types, and whether or not a component has or retains STATE (is a state machine). The types of the INPUT and OUTPUT variables used in the OPERATOR

components must also be defined as TYPE components in the TYPE part of the software base. If a component has states it must be represented by an abstract state machine component, which corresponds to the machine type of operator in PSDL or an Ada package [Ref. 17].

The KEYWORD capability provided by PSDL is also very useful in the classification of Ada components for inclusion in the software base. The KEYWORD is a capability of PSDL that allows augmentation by the user. When used in the component classification and retrieval processes described in later sections, KEYWORDs represent the major domains of interest. The definition of KEYWORDs to be used in a given database of components is analogous to the development of the domain language used in the Draco system mentioned earlier. In combination with the structures that PSDL can describe in its basic form a classification scheme can be derived that will greatly aid the user of the software base.

### 3. The Generalization Lattice Structure

The previous section described some of the more useful information provided in the PSDL specification of a component. The problem now is how to use this information to form a classification scheme for reusable components. The classification scheme used for the software base must meet several requirements to be useful to the prototyper. It must be easily extensible to allow evolutionary growth of the available components. The user should be able to browse available components and select and

retrieve components efficiently. [Ref. 12: p. 70] To support these operations the software base organization must be able to organize components in relatively small, yet distinct classes. Since the goal of the system is to support the overall process of CAPS development this organization should be largely transparent to the user. It must take into consideration the implementation language as well. What follows is a description of a classification scheme that meets these requirements for the OPERATOR type of component.

In order to meet these goals a classification scheme based on categories of information that can be derived from information provided in the Specification portion of the PSDL description is used. A process known as generalization by category is useful in describing the categories of components found in the software base [Ref. 10]. The OPERATOR type of component is particularly suited to this approach. An OPERATOR in PSDL can be a machine or a function, depending on whether it has state. Implementations of OPERATORS in the software base may be generic components or specific pieces of source code representing the major building blocks available in Ada: the package, procedure, function, or task. For these specific implementations we assume that the types that appear in the specifications of these Ada units are restricted to types already known to the software base management system and stored in the SPECIFIC TYPE portion of the software base. A check of the specification can determine if the PSDL component uses these types. If so it belongs to the category of components that

may have exact implementations in the software base. Otherwise, the component is generic. Similarly, if the component has STATE it belongs to the category corresponding to the machine type of PSDL operator.

The categories mentioned above are valid for all operators in the software base. Additionally these categories may be combined to form a lattice structure that further subdivides the OPERATOR side of the hierarchy into four distinct classifications based on only the analysis of the basic PSDL structure. This lattice is shown in Figure 3.3.

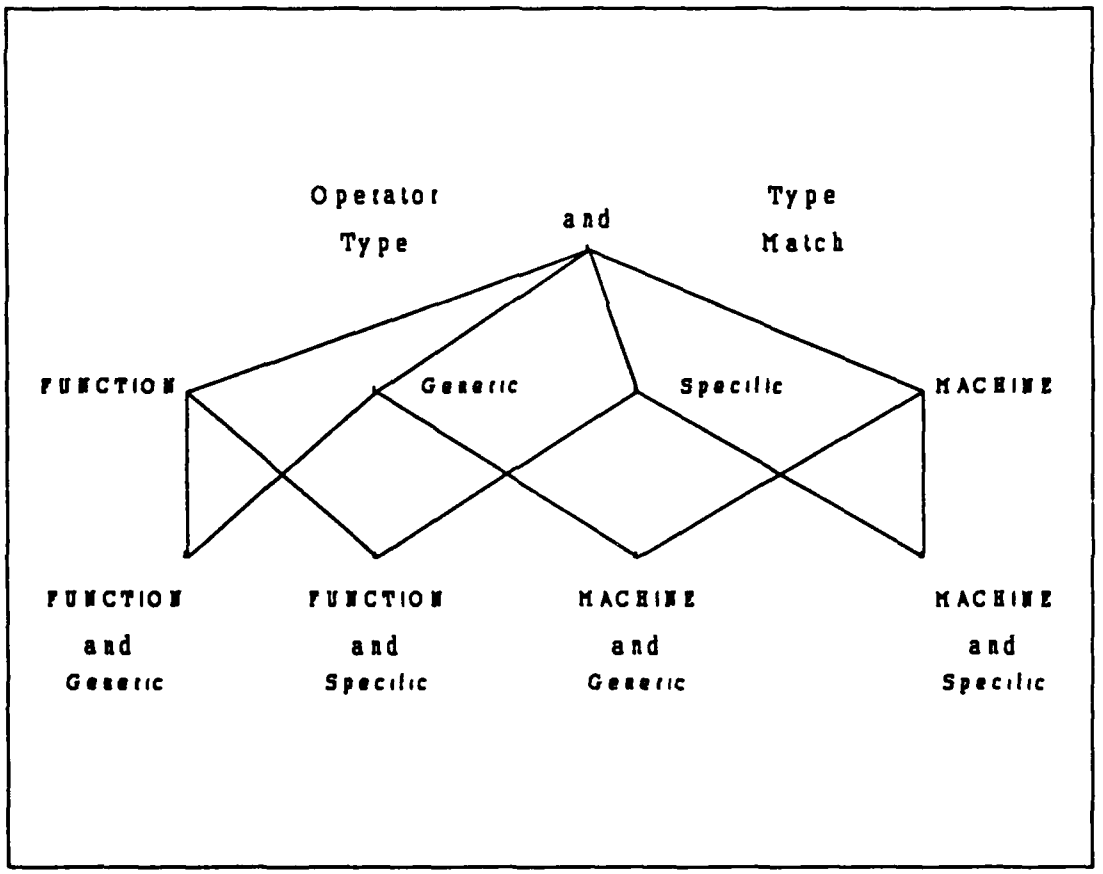



**Figure 3.3 The generalization lattice of categorical properties available in a basic PSDL Specification**

The lattice allows the specialization of OPERATOR components into four classes which can easily be described and differentiated using apriori analysis of the component's PSDL specification. While this is an improvement on the current classification scheme [Ref. 6] there is still the problem of finding a way to further reduce the number of potential components in each class. The lattice provides only a partial solution. However, adding an additional dimension of domain description through the use of the KEYWORD in PSDL can improve classification scheme even further. An example is shown in Figure 3.4:

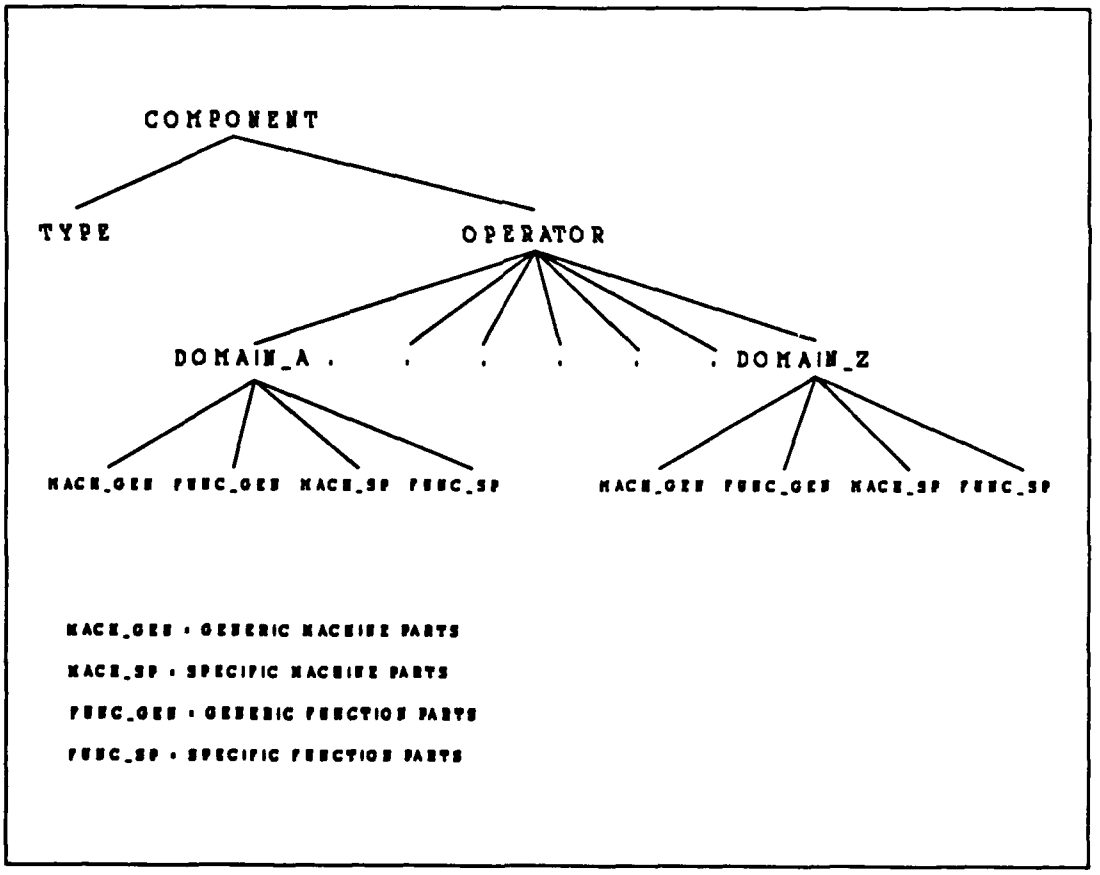



**Figure 3.4 OPERATOR classification with addition of Domain information.**

We see that the number of available classes has been increased significantly from the original single OPERATOR class. This has been done in an orderly fashion by taking advantage of only the basic PSDL language augmented by an arbitrary number of domain related keywords. This type of classification scheme would be adequate for many library type of implementations. But it still does not take full advantage of the capabilities of object-oriented databases. The next section further refines the classification scheme to take advantage of the object-oriented model.

### 4. The class hierarchy of components

In order to effectively organize an object-oriented database more than just the categories of objects must be considered. The previous section showed that PSDL augmented by a domain language of KEYWORDs provides an effective means of organizing domain related components into useful PSDL related objects. However the methods required to incorporate the components into the prototype were not considered. As previously mentioned, one of the advantages of the object-oriented model is the ability to inherit attributes and methods down the class hierarchy. The inheritance of attributes allows the definition of attributes for the entire OPERATOR hierarchy at the OPERATOR class level and the inheritance of the attribute definitions unchanged throughout the OPERATOR side of the hierarchy. The inheritance of methods problem is different.

The software base management system currently uses Ada as the target language for source code portions of the code related objects. One of the parts of the Ada

language that is very supportive of reuse is the generic facility [Refs. 2,16]. Instantiation of Ada generic packages and subprograms allows the multiple reuse of the same source code within the same program. The instantiation method for the generic classes, is different than that used by the specific classes. The problem with implementing these methods using the previously described hierarchy is that there would have to be a specific method attached to each class in the hierarchy. This makes inheritance of methods useless to us in the design of the object-oriented database and greatly increases the programming load. A better solution is to look at the problem of classifying objects as either generic or specific implementations first, and then incorporating the domain information. The modified hierarchy using this method of classification is presented in Figure 3.5. The initial look at this scheme would make it seem that the hierarchy has again been split in half, with the domain names duplicated. While this duplication has occurred, and there is now another level added to the hierarchy, the actual number of classes used to store the OPERATOR components remains the same. The difference is that we can now handle generic and specific source code components with common routines at a much higher level using inheritance to greatly reduce the load of the database programmer. This class hierarchy allows the incorporation of the domain knowledge into the hierarchy of components that can be described by PSDL and also takes into consideration the major classifications of program units in the target implementation language. Its final advantage over previously described methods

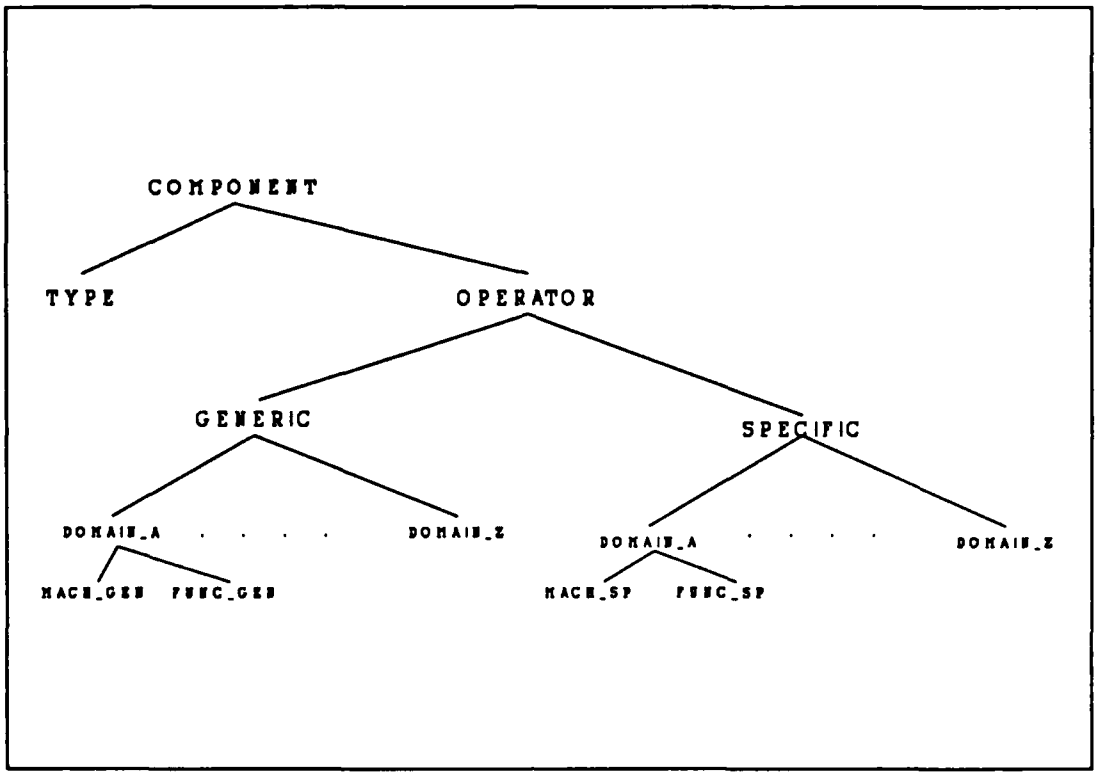



**Figure 3.5 A class hierarchy for the PSDL OPERATOR type.**

is its adaptability to the object-oriented classification model and its ability to support inheritance of both attributes and methods. This is the recommended class hierarchy for the software base. Actual attributes and methods for the objects contained in the database will be described in more detail in the next chapter. However, it can be seen that for a software base of reusable Ada components the use of PSDL with the addition of domain dependent keywords provides the ability to classify components into classes of a much more manageable size than currently exists. The hierarchy must still be accessed in some manner in order to be useful. This is the subject of the next section.

## B. SEARCHING THE OBJECT-ORIENTED DATABASE

The previous section describes a hierarchical structure of an object-oriented database with classes derived from PSDL and domain related keywords. To effectively use the database of components organized in this type of hierarchy, there must be a way to query the database that limits the search to the classes that will most likely contain the needed component. This query is formed from the PDSL specification by transforming the specification using some simple rules. This section describes the process used in the transformation and the role of the RULE objects in this process.

### 1. The Form of the Database Query

The transformation of a PSDL specification into a form that can be used to query the database closely resembles the process used to classify the components into categories. The end result of the transformation is a tuple consisting of information needed select the classes having the best chance of containing the component object as well as the information needed to identify particular objects within those classes. The following information about a component is the minimum required to effectively query the database. Other attributes can be used to find better matches; but, for purposes of this discussion, are not included here:

Inputs: The set of input type names.

Outputs: The set of output type names.

Search_List: The set of classes to be searched.

The input and output sets allow for the determination of an approximate match of individual components to the PSDL SPECIFICATION. The Search_List contains the list of classes that, based on analysis of the specification, have the best chance of containing the needed components. The first two of these attributes can be directly copied from the SPECIFICATION. The Search_List is derived from the transformation of the domain related KEYWORDs contained in the SPECIFICATION. This is done using the RULE class objects related to each KEYWORD. This process is described in the following sections.

### 2. The Structure of the RULE Objects.

There is a tradeoff involved in searching the database between the time involved in searching all possible domains, which maximizes the probability of finding a match, and the search of a limited number of domains, which may not find an existing component. In order to maximize the possibility of finding a match for any given KEYWORD, rules are used. The rules are based on the possible domains that may be associated with a given component object. These domains are determined at the time the reusable component is added to the database. A RULE object is used to store this information. The RULE object also contains information related to the type of OPERATOR component within each subclass of the domain. The general form of a rule attribute is:

**IF** ( TYPE_MATCH, OPERATOR_TYPE, DOMAIN_NAME) **THEN** (Class_List)

The TYPE_MATCH is the result of matching the INPUT and OUTPUT sets of type names with the available types for the given domains in the type side of the component hierarchy. TYPE_MATCH identifies whether the component is in the specific or generic portions of the hierarchy. If TYPE_MATCH does not find a match, then only the generic side of the hierarchy needs to be searched. The OPERATOR_TYPE is the result of finding the reserved word "STATES" in the specification and will identify the component as either a machine of function. The DOMAIN_NAME is a KEYWORD representing one of the domains contained in the hierarchy.

There are rules for each possible subclass of a given domain related KEYWORD. From the lattice structure presented earlier there are four subclasses for each domain KEYWORD: Specific_Machines, Specific_Functions, Generic_Machines, and Generic_Functions. Therefore, there are four possible rules for deriving each domain related Class_List. The attributes representing these rules correspond to the PSDL and Ada related forms of the components. The Specific_Machine rule is used when the path needed is to classes that may contain a possible exact match of a specific Ada package. In this rule's Class_list are the class names of the Specific_Machine classes for the particular KEYWORD being processed and any other related domains determined during component classification and entry into the database. The process of Class_List formation for inclusion in the tuple used to query the database is described next.

## 3. The Transformation Process

The transformation process used here is a pre-processing step. It takes as input the PSDL specification and produces one or more propositions that can be input into a rule base. The rule base contains the list of classes that most likely contain a component that matches all or at least part of the specification. In order to form the proposition(s) the following must be determined:

1. Does the SPECIFICATION contain TYPEs that match those already contained in the Specific portions of the database?
2. Is the component a state machine (Does the SPECIFICATION contain the word STATES)?
3. Which domains have been identified by the user as most likely to contain the components (What are the KEYWORDS)?

The first question is used to determine if the component is a specific component. If the types all match those in the specific side of the TYPE hierarchy, then there is possibly an exact match for this specification contained in one of the specific classes. If there is not a specific type match (~TYPE_MATCH) then limit the search to the generic classes. If the SPECIFICATION contains the word STATES then the component is one of those that represent a state machine. Finally, the domain(s) identified in the SPECIFICATION represent the domains that must be checked for a possible match. The form of the proposition is: (TYPE_MATCH, OPERATOR_TYPE, (DOMAIN_LIST)).

However, the rules used to match the proposition require that each domain be identified separately. So before the rule base is accessed the DOMAIN_LIST is broken into its individual components and the result is one or more propositions of the form:

(TYPE_MATCH, OPERATOR_TYPE, KEYWORD)

There is one proposition for each keyword contained in the specification. These propositions are then asserted in the rule base and matched against existing rules to determine Class_lists for each given domain given the form of the component.

As previously mentioned the rules above are stored in a RULE object in the database. The rule objects are analogous to frames. In each domain or frame of reference the possible forms of components are stored along with the possible classes that may contain matching components. Therefore based on the discussion of the lattice of classification there are four rules for each domain. The forms are:

1. The Generic Machine rule:

   (~TYPE_MATCH, MACHINE, KEYWORD) ->

   (Generic_Machine_Class_List)

2. The Specific Machine rule:

   (TYPE_MATCH, MACHINE, KEYWORD) ->

   (Specific_Machine_Class_List)

3. The Generic Function rule:

   (~TYPE_MATCH, FUNCTION, KEYWORD) ->

   (Generic_Function_Class_List)

4. The specific function rule:

   (TYPE_MATCH, FUNCTION, KEYWORD) ->

   (Specific_Function_Class_List)

The Search_List is formed from the transformation of each of the domain related propositions using the union of all the Class_Lists. In short the transformation process can be described by the following steps:

1. Convert the SPECIFICATION into a basic proposition containing the results of the Type_Match and Operator_Type functions and the Domain_List.

2. For each KEYWORD form a proposition in the form of a rule antecedent.

3. While there are still propositions to process:
   Match to Rule_base to get Class_list
   Perform Union with previous Class_list

4. Return final Class_List. This is the Search_List.

The final returned Class_list identifies the classes to search. It includes all the classes identified by the user as well as those identified in the classification process of the individual objects as they were added to the database and included in the rules. It is this

list that is used control the search of the object-oriented database. An example of this process is shown in the next section.

## C. A SAMPLE USE OF THE CLASSIFICATION SCHEME

The previous sections have shown how PSDL could be used as a classification language for storing reusable components in an object oriented database. In this section an example is presented using the component description and implementation shown in Figures 3.1 and 3.2 and the sample taxonomy shown in Figure 2.1. Component classification, including a description of the associated RULE object, performed by the database administrator at the time the component is loaded, and the derivation of the Search_List, required for search and retrieval operations, are shown for the sample specification and implementation.

### 1. Classification of the Reusable Component.

The component shown in Figure 3.2 is an example of a specific function in terms of PSDL. The use of the taxonomy previously described would place this function into one of three domain related classes: Guidance and Control, Asynchronous Control, or Specific Equipment Interface. The classification selected is a function of the database administrator using input from the domain analysts. But, regardless of the selected category, the cross-referencing information is built into the associated RULE object for the selected class. In this case the object is classified as an instance of the class Specific_Function_Guidance_and Control. The Specific Function attribute of the RULE

object corresponding to Guidance_and_Control is updated to reflect the other possible domains. The form of the specific function attribute for guidance and control is now:

```
(TYPE_MATCH,FUNCTION,(Guidance_and_Control)) ->
                (Guidance_and_Control_Function_SP,
                 Asynchronous_Control_Function_SP,
                 Specific_Equipment_Interface_Function_SP)
```

This process is done for each component when added to the database. For purposes of illustration we assume that the Asynchronous Control and Specific Equipment Interface rules have the specific function attributes shown next.

The Specific_Function rule for Asynchronous Control:

```
(TYPE_MATCH,FUNCTION,(Asynchronous_Control)) ->
                (Asynchronous_Control_Function_SP,
                 General_Purpose_Equipment_Interface_Function_SP,
                 Guidance_and_Control_Function_SP,
                 Non_Guidance_Control_Function_SP)
```

The Specific_Function rule for Specific_Equipment_Interface:

(TYPE_MATCH,FUNCTION,(Specific_Equipment_Interface)) ->

(Specific_Equipment_Interface_Function_SP,

General_Purpose_Equipment_Interface_Function_SP,

Guidance_and_Control_Function_SP,

Communications_Function_SP,

Navigation_Function_SP,

Non_Guidance_Control_Function_SP)

The final part of the classification process involves updating the TYPE side of the hierarchy to include objects referencing the INPUT and OUTPUT types used by this OPERATOR component. Once all these operations have been accomplished the component is stored. The method of retrieval is described in the next section.

### 2. Retrieval of Components from the Software Base.

Retrieval operations are triggered by the CAPS user during the process of specifying a PSDL operator. The input into this process is the PSDL description augmented with KEYWORDS. Figure 3.6 shows an augmented PSDL specification for the controller_start_up OPERATOR shown previously.

```
PSDL DESCRIPTION

OPERATOR controller_start_up
SPECIFICATION
    INPUT control_panel: real, sensor_input: real
    OUTPUT missile_control: real
    MAXIMUM EXECUTION TIME mu
    BY REQUIREMENTS control_panel_max
KEYWORDS Guidance_Control, Specific_Equipment_Interface
DESCRIPTION
{ Extracts the control panel input and sensor input and
  uses them to calculate the control signal for the missile.
}
END
```

**Figure 3.6 Sample Augmented PSDL Component Description.**

The addition of the KEYWORDS allows the component description to be processed using the method described in this chapter. The Types would be selected from the domains referenced in the component description and used to determine the TYPE_MATCH part of the proposition. The examination of the description shows no STATE variables, which sets the second part of the proposition to FUNCTION. The domain list is the list of KEYWORDS contained in the descripti In this case there are two members contained in the domain list. This results in two assertions to process and one Union operation of the class lists returned from the rule base. The Search_List derived from processing this component is:

(Guidance_and_Control_Function_SP,

Asynchronous_Control_Function_SP,

Specific_Equipment_Interface_Function_SP,

General_Purpose_Equipment_Interface_Function_SP,

Communications_Function_SP,

Navigation_Function_SP,

Non_Guidance_Control_Function_SP)

This list and the PSDL description are then input into the object-oriented database to continue the attempt to match this specification with a reusable component. The list limits the classes to search and the PSDL description provides the information needed to conduct the more detailed operations required for selection of components from the database and incorporation of the component into the prototype.The inclusion of the rules derived from the classification process insures that a more complete search will be done and solves many of the problems associated with cross referencing. A system that implements this process is the goal of the design effort. A structure that supports this effort is described in the next chapter.

# IV. CONCEPTUAL DESIGN OF THE SOFTWARE BASE

In this section an extended conceptual design of the software base is presented. The goal is to integrate the concepts presented earlier in the discussion of domain analysis, rule based systems, and object oriented programming and databases, into the design of a subsystem that extends the current approach used in CAPS. The objective of the design presented here is to identify the major objects that are required to implement this subsystem. Additionally, for the object-oriented database component, the classes of objects and the methods required of these classes to support the ultimate goal of integration of reusable components into a CAPS prototype as well as the basic functions required for building and maintaining this database, are described.

The design presented here is generic in nature. It describes a database that represents domains by defining small classes of reusable component objects using concepts from the domain analysis and the PSDL language and organizing them into a hierarchy. It also stores rules for each domain in a specialized object class. The search of this database is aided by the transformation of the specification prior to querying the database as described in Chapter III. The results of the transformation and application of rules is the identification of the classes to search in the database.

The discussion begins with the description of the reusable software base and its interface to the CAPS system. This is followed by an explanation of the major subsystems required to integrate reuse into the system. This description will be presented following the object-oriented analysis methodology described in Chapter II. Finally the major forms of the objects used to store reusable components and the specialized class used to store the rules is presented. The attributes and methods that each of the major object classes need to support the required system functions is described in sections detailing each major object class.

## A. THE REUSABLE SOFTWARE BASE SYSTEM ARCHITECTURE

Chapter II presents an overview of the CAPS architecture. The Software Base Management System component contains two supporting databases: a database of reusable software components and a design database. The component we are concerned with here is the reusable software base. For purposes of this discussion the design database can be considered another part of the CAPS environment described in Figure 4.1. This is reasonable because its interface with the software base of reusable components is the same as that of the other components of CAPS. Other parts of the environment that interface with the reusable software component subsystem are the syntax directed editor and the translator. In all cases the basis of the interface is a PSDL specification.

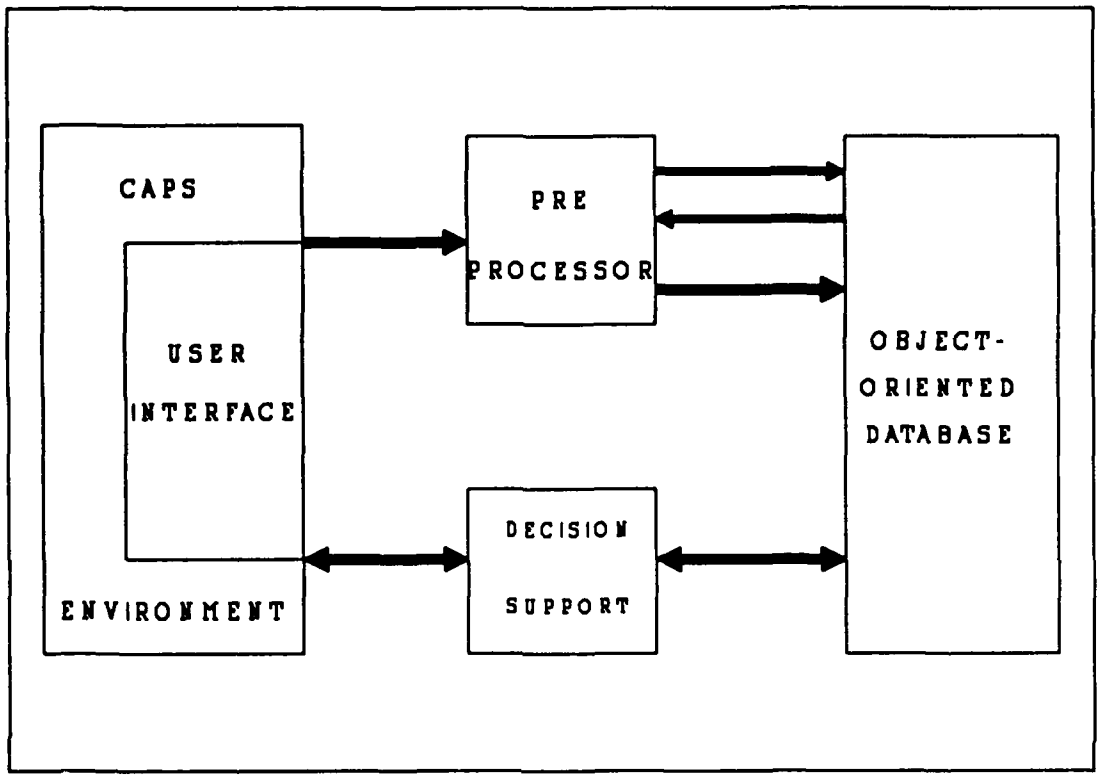



**Figure 4.1 The Software Base System Architecture**

The major parts of the software base of reusable components subsystem are the pre-processor, object-oriented database and the decision support modules. These are described in the following sections.

### 1. The Preprocessor

The preprocessor consists of those parts needed to determine the Search_List of classes likely to contain reusable components. Included in this module are a parser that does the transformation of the PSDL specification into a proposition and the rule

base. The rule base is formed by retrieving selected rule objects from the rule class in the object-oriented database. Specific rule objects are determined by the Domain_List formed during the transformation of the specification. Associated methods contained in the rule class are covered in the description of the rule class later in this chapter.

### 2. The Decision Support Module

This module performs the functions of object selection and incorporation of the reusable software component into the prototype. There are a number of alternatives that can be considered in the implementation of this module. These include the implementation of a manual means of browsing components identified during the iteration through the classes, the selection through automated analysis of other parts of the specification or, more likely, a combination of the two. Research in this area is currently underway. The proposed structure is designed to support this research.

### 3. The Object-Oriented Database

This module contains the actual components with the attributes and methods needed to support addition, deletion, modification, and update of rules used to support the search and selection of components. Note that methods must exist that allow the database to interact with both of the other modules. The structure of the database and the attributes and methods are described in the next section.

## B. THE STRUCTURE OF THE OBJECT-ORIENTED DATABASE

There are three main types of objects that are needed to implement reuse in the CAPS software base of reusable components. Two are already supported to a limited extent - TYPES and OPERATORS. These classes of components represent the major constructs used in PSDL. However, the previous design [Ref. 6] is oriented to storing and retrieving components which are already well known by the user. To properly use the software base, the user must know the exact name of the component to retrieve it. The approach taken in this design is to create a class hierarchy to support a search based on partial description of the object as described in Chapter III. To do this the TYPE side of the class hierarchy is expanded to include subclasses associated with the application domains that they are used in. The OPERATOR class is expanded in a similar manner incorporating domain related classification into the expanded hierarchy well as information on the form of the component from the target implementation language. To support search operations an additional object class, containing RULE objects, is added. The RULE class contains the simple rules used to support search and retrieval operations. A top-level description of the extended database hierarchy is shown in Figure 4.2. All classes in an object-oriented database are defined as subclasses of some basic object class that provides the basic attributes and methods needed to provide basic database operations such as add, delete, modify etc. These attributes and methods are inherited and may be modified further down the class hierarchy. We assume that this is the case in this design

is the case in this design and use the class labeled OBJECT as the base class. The following sections further elaborate the top-level classes attached to the base class and describe the required attributes and methods of these types of objects. Many of these attributes and methods are inherited from higher levels, but modified to provide additional capability.

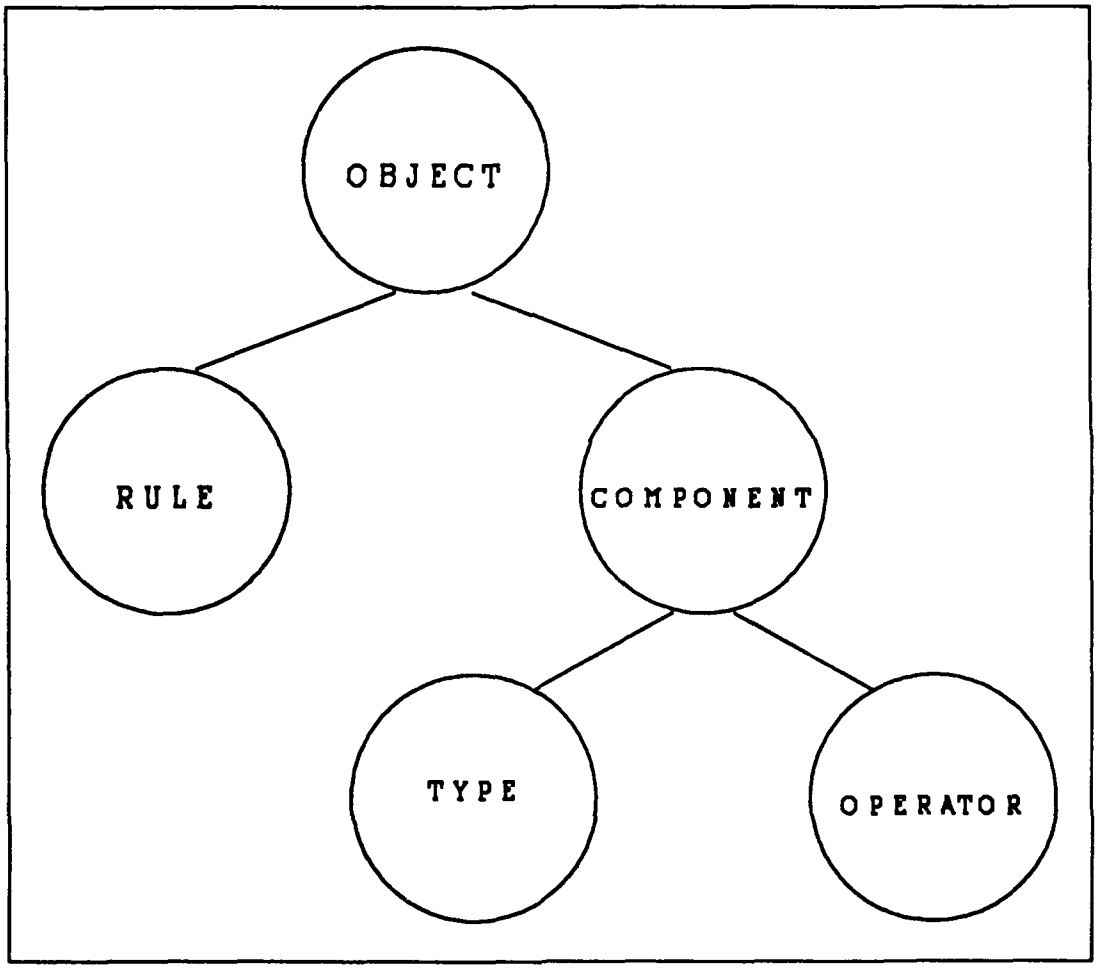



**Figure 4.2 The Class Hierarchy**

### 1. The TYPE Objects

The TYPE hierarchy, or subtree of the component hierarchy, is composed of objects that support the Ada type definitions used in the component objects found in the OPERATOR portion of the database. From the domain analysis and commonality studies, types used in the implementation of OPERATORs and associated with the application domains in the software base are identified. It is this association that is used to form the hierarchy of types. An example of such a hierarchy is shown in Figure 4.3.

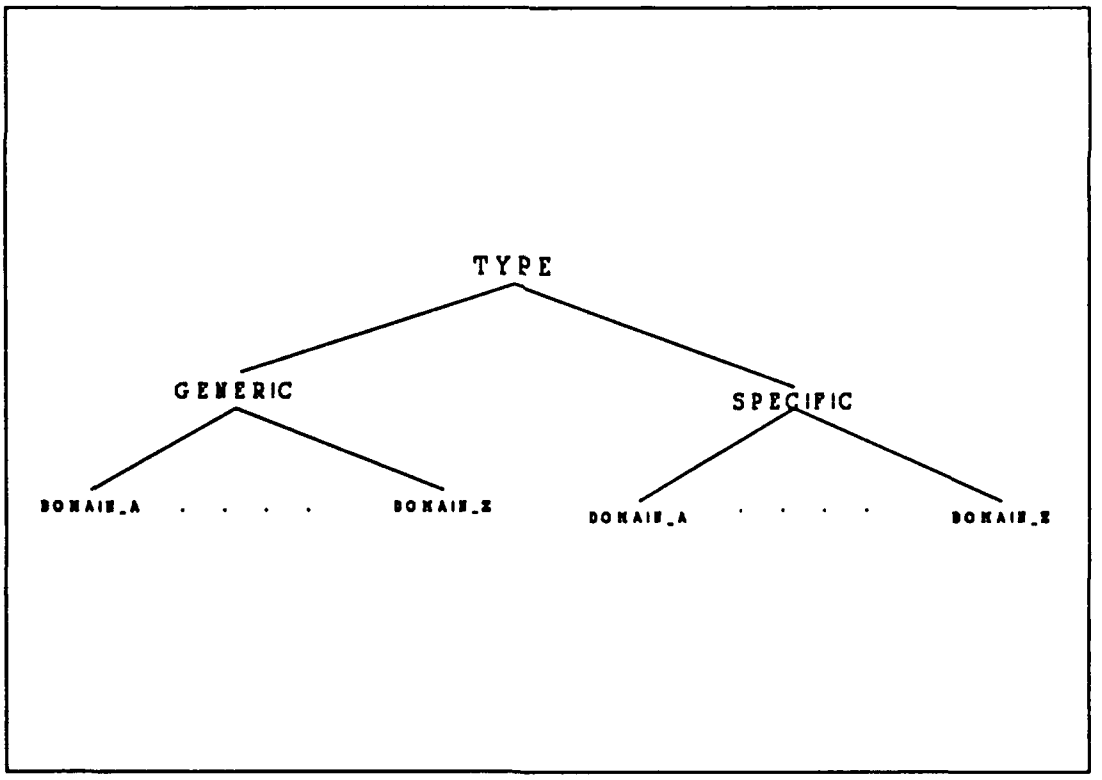



**Figure 4.3 The TYPE hierarchy.**

Each of the actual objects is basically the same. The classification scheme is merely a means of organizing the objects into a hierarchy based on the application domains. There are two advantages to this separation. First, the operations and the attributes needed by all TYPEs can be described once at a higher level and inherited by each of the domain dependent subclasses. The operations needed to support retrieval can also be stored at a higher level and invoked by sending the message to the object class at the domain level. The operation is inherited and applied to the referenced class. Second, capabilities to provide indexing information are often provided for classes defined in an object-oriented database at the class level. The subdivision of TYPE into the smaller domain related classes allows us to take advantage of this capability in the actual implementation.

The following are the required attributes and methods for the TYPE side of the hierarchy in the Software Base:

Attributes:

Name: A string representing the Ada Type Name. Inherited from the Component Class definition.

Operator_List: The set of associated OPERATORs in the same domain using this type. Supports Add and Delete Operations in the OPERATOR hierarchy. Can also be used to trigger delete operations for individual instances of TYPE objects.

Ada_Text: The actual text of the type declaration in Ada Syntax. Definition inherited from the Component class.

Description: Information which describes the type and may be useful while browsing the database.

PSDL Specification: The PSDL specification for the TYPE. PSDL Descriptions are required for more detailed analysis.

Methods:

Add_Type: Supports the addition of components to the software base by updating the types contained in the domain of the added component. Used in conjunction with Add_Operator method in the OPERATOR side of the hierarchy. Types are only added when required to support OPERATORS.

Delete_Type: Supports deletion of types, as required, from the software base. Deletion is done when all OPERATORS contained in the Operator_List have been eliminated. Done in conjunction with, and triggered by the deletion of OPERATOR components.

Get_type: Used to get the Ada text representation of the type. May be used in display operations or operations to incorporate class into program. Displays the Ada_Text and description to the screen.

Form_type_list: Used to reference the class hierarchy for class names contained in the Domain_list formed as part of the transformation process of the PSDL specification. Returns the objects found in each of the classes. Used in transformation process of PSDL SPECIFICATION to proposition. Supports the TYPE_MATCH operation described earlier.

This description includes the minimum required set of operations and attributes. This section of the hierarchy has a great deal of future potential. For example, an area which may benefit from the reuse of class information is the definition of data streams in the graphical editor. However, since the main problem addressed in this thesis is retrieval of the OPERATOR type objects, the discussion of the TYPE hierarchy and

objects is limited to the attributes and methods needed to support the maintenance of the database and retrieval of OPERATORS.

### 2. The OPERATOR Objects.

The OPERATOR objects represent and encapsulate the major reusable components contained in the software base. It is here that the actual code of Ada procedures, functions, and packages is stored. Galik provides a good description of the major portions of the OPERATOR class used for each of the objects [Ref. 6]. However, since he provides only one OPERATOR class, he does not address one of the strengths of object-oriented databases. This is the ability to successively specialize objects in a hierarchical fashion. Figure 4.4 shows an example of this expanded hierarchy. The specialization of object classes by functional areas defined during domain analysis, combined with the major forms of OPERATORS that can be represented in PSDL results in an expanded hierarchy that can be searched more efficiently.

The requirements of the prototyping system make it necessary to store more than just source code in the component objects. Attributes such as timing characteristics and other descriptive information must be put into separate portions of the object to assist in the evaluation and selection process. There is also a need to store information needed to cross reference objects and classes. This is done for two reasons. The first is to insure a complete search is conducted during retrieval operations. The second is to insure that all components needed to properly use the referenced component are also found. Two

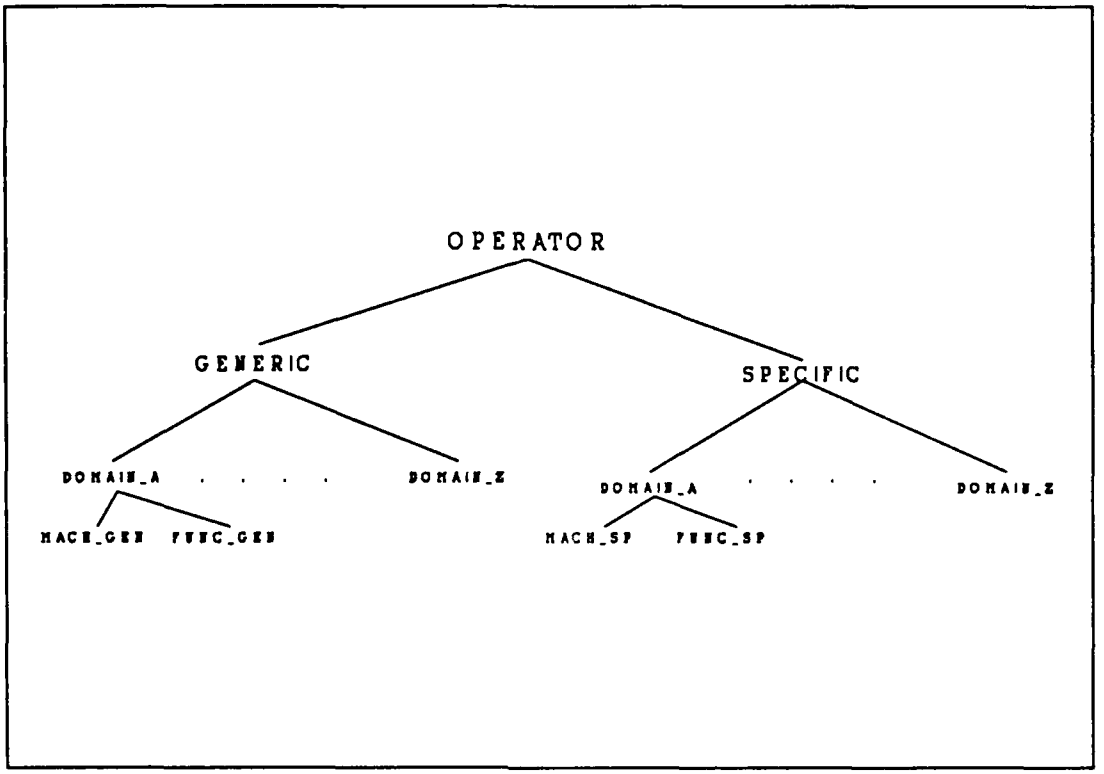



**Figure 4.4 The OPERATOR hierarchy.**

attributes are defined to provide the additional information needed to support these operations. The first of these is a "Withing" list which is used to locate support modules needed by the component to be properly used in a program. This directly corresponds to the With of the Ada language. The other main reference attribute is a set of related classes. This is required because of the possibility of overlap between the functional domains. Any given component will be classified into one domain related class. However some may have applicability to more than one class. A master path list containing the names of all domains related to the members of each class is maintained

in the RULE object section of the database. The attributes and methods needed for all objects at the base level of the OPERATOR section of the hierarchy are:

Attributes:

Name: The name of the component. This name is the same as that of the Ada component stored in the text portion of the object.

Inputs: The set of variable and type names used in the implementation of the object.

Outputs: The set of variable and type names used in the implementation of the object.

With: The set of objects used to fully implement the operations of the component object using the Ada "With" capability.

Path_list: The set of related classes that may also contain modules with the same or related functionality as the referenced component.

Requirements_list: Used to describe the functionality of the component. In PSDL terminology the "BY REQUIREMENT" phrases would be stored in this construct. This attribute is not required for the search or retrieval process described here. May be useful in the Decision Support module for more detailed analysis of the individual component objects.

Description: Text describing the module. In the form of the PSDL SPECIFICATION with documentation added as natural language text.

Source: The Ada code of the specification and body of the component.

Timing: Maximum Execution Time. The timing characteristics may also be detailed as separate attributes.

Methods:

Get_the_Object: Needed to support further processing or display of the object for review by the user.

Display_Description: Displays the Description text only for evaluation by the user.

Add_Operator: Used by the software base administrator or librarian to add components of type OPERATOR to the database and update associated TYPE and RULE objects. Refines the basic add operation defined at higher levels in the hierarchy.

Modify_Operator: Needed to change information (attributes) of the individual objects. Used by the system administrator or librarian. Refines modify operations inherited.

Delete_Component: Used to delete components no longer needed and update associated TYPE and RULE objects. Also a function of the database administrator or librarian. Updates the Operator_List on the TYPE side of the hierarchy and may cause the method to delete a TYPE to be activated. Refines inherited methods to delete objects.

Write_Component: Refines inherited write methods inherited from the Component class. Used to add selected objects to the prototype in progress. As CAPS is currently implemented this method would open and write to the file the source code for the selected object. This method would vary depending on the location of the selected object in the hierarchy. There are two main options:

Specific OPERATORS: This method will copy the text of the source attributes.

Generic OPERATORS: The instantiation will be done using the Write_Component method found in the generic side of the OPERATOR class hierarchy. This is an example of overloading of method names allowed in object-oriented languages as explained in Chapter II.

Get_Axiom: This method is used for support of more detailed matching operations currently being researched. The AXIOM is a CAPS construct used to more fully describe in formal terms the OPERATORS.

The attributes and methods presented here are considered the minimum required to support the process of matching PSDL specifications to reusable software components described in this thesis. There are many other requirements that must be met as part of the software development process. Some of these include version control, for the objects contained in the database, and documentation requirements supporting the requirements definition and design processes CAPS is designed to support. These are just two of the areas that may be expanded in future research.

### 3. The RULE Objects.

Objects contained in this class are used to assist in the selection of classes in the OPERATOR portion of the database to search based on the query formed by transformation of the PSDL specification. The goal of the rule base is identify the "most likely" class(es) to search in the database. The RULE class contains simple first order logic representations of the Search_Lists (or maybe more appropriately class lists) that match the pattern of the input proposition formed by transforming the PSDL specification into a proposition that can be used by the rule base. The end result of the processing done by the rule base, as described in Chapter III, is a master search list of potential cl. es which may contain the required objects. This list used to guide the subsequent search of the database.

The rules themselves are built by using the basic features expected from any PSDL component on the left hand side of the rule (the antecedent) and the related class lists

found in the objects of each domain's specific and generic classes on the right hand side. As new objects are added to each class, the related class list is compared to that already existing for the class. If some new element is found in the class list it is added to that portion of the corresponding rule object's list.

The description of the class RULE objects follows:

Attributes:

Name: The domain name of the domain being represented by the rules stored in this object.

Gen_Machine_Path: Rule representing the path (class list) to be searched if the antecedent results in limiting the search of related classes to only generic classes. The generic machine rule.

Gen_Function_Path: Rule representing the path (class list) to be searched if the antecedent results in limiting the search of related classes to only generic classes that do not contain machines. The generic function rule.

Specific_Machine_Path: The comprehensive search path for the named starting domain. This path list contains all of the classes including generic and specific for the named domain of interest. The specific rule for finding machine components.

Specific_Function_Path: The comprehensive search path for the function components. This path list contains all of the classes including generic and specific for the function components. The specific rule for finding function components.

Methods:

Get_Rules (Domain_list): Method to build the rule base from the identified domain lists. Called by the Preprocessor module. Rules returned are used to get the class list that is searched.

Add_Rule: Method to create instances of the class rule when creating new domains of objects in the database. Inherited from the OBJECT class and triggered by the creation of a new class by the database administrator or librarian. Initially it would be empty. As objects were added it would be updated using the modify rule method below.

Modify_Rule: (Domain_list): For an existing instance this method takes a modified list of domains and modifies the Gen_Path and Specific_Path attributes to reflect changes in the Path_list of the domain related class (found in the OPERATOR hierarchy). This method would be used by each OPERATOR create process. Performs a comparison of the new domain list included with the newly created or modified object and would perform an "OR" or Union type operation on the lists to insure any new classes were added to the path lists.

Delete_Rule: When an entire domain related group of classes are no longer required the rule object related to this domain would have to be deleted. This method would be inherited from the basic OBJECT class and triggered by maintenance programs used by the database administrators.

The processing of the rules is part of the preprocessing done prior to the actual search of the OPERATOR portion of the database. It is, therefore, important to decide at what point in the prototyping session the rule base gets built. The major question to be answered is whether or not to build a comprehensive rule base at the beginning of each session, limit the operator of the CAPS to some subset of domains per session, build a new rule base for every query, or start with a rule base formed from a given set of domains and continue to add rules as more domains become active during the session. The approach in defining the attributes and methods used in the RULE class definition is to remain flexible enough to support any of these approaches.

## C. SUPPORT OBJECT CLASSES AND COMPOSITION OF OBJECTS

One of the more important capabilities of the object-oriented development model and object-oriented databases in particular has not yet been considered. When designing the detailed implementation it will be necessary to provide the user capabilities to display the object or perform manipulation on various attributes. Rather than define methods for manipulation of each part of an object, we can take advantage of system supplied object classes that can be composed into parts of the specialized objects we are interested in. In Object-oriented databases there is often an environment already available built on to the basic Object definition and inherited by, or in some way made accessible to all subclasses of object. This is the case in the Ontos system [Ref. 19], the projected implementation system for the expanded software base described in this thesis. The next chapter describes some of the more useful capabilities this system may provide in support of the construction of the expanded Software Base Management System.

# V. IMPLEMENTATION ISSUES

The previous chapter describes the needed objects and organization of the class hierarchy to support selection of reusable software components using simple rules to determine where to start the search. The major goal addressed in the description of the objects was to identify the structures needed to support the search process. The overall goal of the pre-processor and database components of the system described here is to narrow the number of candidate objects and present a best qualified set of candidates to the decision support component of the system for more detailed analysis. To insure a more complete search of the database, cross-referencing information needed to support this search is stored as one attribute of the major types of reusable component objects. Additional attributes needed to support user interface, documentation, and more detailed matching operations are not included in this description. Supporting object classes included in the implementation environment are a means of further abstracting portions of the custom object. The design presented is based on exploiting the object-oriented model and the concepts of domain analysis for a generic domain. Chapter IV avoids specific issues related to the actual building of this system. In this chapter some of these issues are addressed.

## A. THE ONTOS OBJECT-ORIENTED DATABASE SYSTEM

Ontos is the system currently projected for the implementation of the Software Base Management Component of CAPS [Ref. 19]. It is the second effort from the developers of the VBase OODBMS system currently used in CAPS [Ref. 6]. There are several major improvements that are included in this product. It is designed to provide direct object access from C++ applications and to provide much better performance than VBase. Plans are to improve the overall environment to ultimately provide a graphical oriented environment to support graphical schema definition and modification, menu, form and report generation, query by example, database browser, and simple applications generation [Ref. 19: p. 5]. It is one of the first commercially available object databases supporting the C++ programming language.

The classes provided by Ontos contain many of the support objects previously mentioned as important to the implementation of the expanded software base system. The built-in data types provided by the interface language, Aggregate Classes used to collect groups of objects, and an Iterator Class which can be used to step through the attributes of individual objects or Aggregates of objects are probably the most important. These and other support classes and functions are contained in the Client Library provided by Ontos. [Ref. 19: pp. 51-96]

The base classes of the Ontos system are the CleanupObj, Entity, and Object. CleanupObj provides the Iterator capabilities useful in stepping through the attributes of

individual objects or objects collected in aggregates. Entity provides uniform reference semantics for classes and C++ primitives. Primitives represent the C++ built-in types. Object provides persistent object definition for objects stored in the database. The schema definition, directory capabilities, and Aggregate classes mentioned previously as important to the implementation of the search and retrieval methods described in this thesis are also derived from this class. Classes for defining the operations and attributes to be included in objects of the customized classes are included here as well. The Object class provides operations that can be used in the definition of more complex methods at different levels in the hierarchy. For example, modification of the New-Instance constructor function for the OPERATOR class described in the previous section to allow the update of the RULE class object associated with this class could be done at the OPERATOR level and inherited to each of the subclasses. [Ref. 19: pp. 50-96]

In summary the Ontos object database has adequate capability to quickly implement the database needed to support the reuse system described in this thesis. It also has the capability, using the Set class and Iterators, to support operations that further narrow the number of possible objects selected as candidates for detailed matching operations performed by the decision support module described earlier. This would further aid the decision process this system is designed to support. Implementation of a system using Ontos is required to test many of these features.

## B. PROTOTYPING OF THE EXPERT SYSTEM COMPONENTS

Two of the three components described in the architecture of the expanded software base management system are based on expert systems. The Ontos system described previously provides the capability for providing the needed information to support both the pre-processing and decision support components of the software base management system described here. The C and C++ programming languages are useful for building higher performance expert systems than those written in Lisp and Prolog. However this conversion is normally done only after the concepts are proven and performance becomes the primary consideration. There is a general method used in expert systems development. In the systems surveyed as part of this thesis the expert systems components were all developed as follows: [Refs. 1,3,19-21]:

1. Problem Assessment (Domain Analysis)
2. Prototyping and Knowledge Base Construction
3. Software Engineering and Program conversion

The expert system components were not converted until the knowledge and rules needed for the problem domain were reasonably firm. It is then that performance becomes the major issue. The methodology presented in this thesis and the process of domain language development for use in CAPS would benefit greatly from the construction of a prototype expert system which could be continually improved. New rules and expanded

frames of reference could still be incorporated into the RULE objects in the OODBMS and used by a system based on the interface language of the database.

## C. THE REUSABLE SOFTWARE COMPONENTS

As previously noted the reusable components in a given system may have many different definitions. In CAPS the system is relatively flexible, allowing systems to be specified at various levels of abstraction. Subsystems representing high level systems and many lines of code can be stored in the same database providing low-level functions. This makes it possible to reuse code written at varying levels of abstraction and from a variety of sources.

The CAMP project also took an approach similar to that used in CAPS for development of a set of parts (reusable components) [Refs. 22-25] to support the missile embedded systems environment. Many of the modules in CAMP represent embedded system functions that are applicable across a wide range of embedded systems applications. Not all of CAMP can be used, and because of the composition strategy used in CAMP there are several components that may require restructuring to be used in CAPS. This is, however, the best source of real components for embedded systems use that was found during this research. Another source which has many components which may prove useful is the Ada Software Repository [Ref 26]. Many mathematical and data structure parts can be found in this system. However, the organization of this system does not make it easy to find any particular component.

## D. DEVELOPMENT STRATEGY

The preceding sections have mentioned many of the areas that must be considered in the development of the proposed system. The normal development of a domain related programming library or reusable component database relies on the development of domain dependent systems that cannot be easily expanded or reconfigured to add new classes to the system. The development of flexible database support for a specific application domain is based on having the ability to add, delete or reorganize entire classes in the database. The system described here allows a modular development strategy. It is possible to reconfigure the domain language by identifying the domain related KEYWORDS from those already in the database adding those needed to describe new classes of components. This development strategy allows the development of new application specific rule bases using already existing classes and reuse of existing RULES.

# VI. CONCLUSIONS AND RECOMMENDATIONS

## A. SUMMARY

The CAPS environment is a collection of tools which allow the development of systems using a methodology based on iterative decomposition and refinement of programs supporting hard real-time requirements. The CAPS system uses the Prototyping System Description Language (PSDL) to support this development process. Descriptions of subsystems or program modules written in PSDL may be further decomposed, translated or incorporated into the prototype in the form of a reusable software component. The subsystem used to manage reuse in CAPS is the software base management system. This subsystem does the identifies and supports incorporation of reusable software components into the prototype. This is done through analysis of the PSDL description, search of the database of reusable components for potential matches, and detailed matching operations to determine if an acceptable component exists.

The previous organization of the software base of reusable components is not adequate to perform its intended function effectively or efficiently. In this thesis a revised and expanded structure for the software base was presented that takes advantage of the object-oriented model and characteristics of object-oriented databases to allow the classification of reusable components into classes which are more representative of the given application domain. As part of this process PSDL was evaluated for its

effectiveness as a classification language. PSDL in its basic form is adequate for the classification of OPERATORS and TYPES into the basic forms used during prototyping sessions. PSDL augmented with a few domain oriented keywords which are allowed in the language definition further improves the ability of the developers to incorporate domain knowledge into the reuse system. Using PSDL augmented by keywords allows the developers of prototyping systems supporting particular areas to build reusable component databases which represent the major objects or subsystems and functions for identified domains of interest.

The use of rules to support retrieval of components from the database was also explored. Again, the focus was on evaluation of rules which could be described using parts of the specification portion of a PSDL description. The goal of the rule base is to identify the most likely class or classes to search for components. The rules are stored in specialized objects in the database and provide cross referencing information needed to insure a more complete search of the database for the desired component. The PSDL description is transformed into one or more propositions that can be asserted in the rule base to get a list of the classes most likely to contain the component. This list along with the description form the input to the object-oriented database and provide needed information to perform more detailed matching operations needed to select components for incorporation into the prototype.

During the course of this research several areas of commonality became apparent in research efforts aimed at reuse of software. All were using some sort of rule base to determine the transformation or composition strategies to be used. All were oriented toward the transformation of some higher order language into a form that could be used to either find components stored in a database or generate components in a target language. All approaches used some form of pattern matching to rate and select components. None were able to totally automate the process for any but the smallest problems. But, in examining the composition based systems, there were as many different composition methods as there were systems. All the other subsystem designs were driven by the design of the composition system.

There are three major activities involved in the selection of reusable components from the software base: transformation of the specification into a form that can be used to query the database; search of the database and retrieval of candidates for analysis; final detailed analysis of the component for possible incorporation into the program or in this case prototype. Follow-on research in this area should focus on implementation of these functions as separate subsystems either within or as interfaces to the object-oriented database of reusable components.

## B. RECOMMENDATIONS FOR FURTHER RESEARCH

This thesis identifies three major subsystems which are involved in the implementation of an effective mechanism for incorporating reuse into CAPS. None of

these methods are implemented. The following recommendations are made for research based on these areas:

- Preprocessor and rule base: Implementation of the rule base based on the structure of the rules described in this thesis. Expansion of the rule base through addition to the domain language of concepts allowing higher levels of abstraction. Implementation of a parser to transform the PSDL description into a proposition and a rule base constructor method to retrieve the rule objects from the database and form the rule base.

- The Object-Oriented Database: Design and implementation of a class hierarchy based on the results of a domain analysis using the method of object classification described in this thesis. An existing domain analysis and the components designed to support a composition based system is part of the CAMP project. Many of the CAMP components are applicable to a number of application domains in the area of embedded hard real-time systems. Other areas of interest include: evaluation of the SQL capability provided by Ontos to further narrow the number of candidate objects prior to detailed matching operations; the storage of documentation information in the database and the automatic generation of documentation required to support the requirements definition for the prototype as well as parts catalogs describing the components stored in the system.

- The Decision Support Subsystem: There is currently ongoing research in the area of detailed matching of specifications in order to do the rating and selection of a component from a set of candidate components. However, it is unlikely that the user would be totally excluded from this process. Research on Browsing and implementation of a browser within this subsystem is required to allow the user to more effectively participate in the final selection of the component object.

## C. CONCLUSIONS

The Computer Aided Prototyping System and the Prototype System Description Language provide the capability to effectively integrate reusable software components as part of a software system prototype. PSDL supports the concepts of object-oriented

analysis and can be used as the basis for both classification of reusable components and development of rule based systems to aid in component retrieval. Object-Oriented database technology, while still relatively new, offers the capability to support this process in a way that allows the reuse system to take advantage of the strengths of the prototyping language as well as the target implementation language. Domain analysis of the application domain is also important to the process of building the database to support prototyping. This thesis proposes a method that can be used to take advantage of all of these areas in the construction of an expanded system to support the reuse goals of CAPS. Implementation of the structure and methods described here will provide a basis for the continued improvement of the reuse capability in the CAPS environment.

# APPENDIX A. THE COMMON ADA MISSILE PARTS PROJECT

The Common Ada Missile Parts Project (CAMP) is an ongoing effort started in 1984 by the Air Force Armaments Test Laboratory and performed by McDonnell Douglas Astronautics Company with the goal of implementing software reuse in the development of missile related embedded systems. This project was studied in detail during research for this thesis for the following reasons:

1. It is an effort to build a real system for software generation in the embedded systems environment.

2. It applies many of the concepts of reusable software engineering described in this thesis as required to support the construction of an expanded reuse system in CAPS.

3. Both CAMP and CAPS use Ada as the target implementation language in the application domain of a hard real-time systems. Many of the reusable software components (parts) developed for CAMP may be useful as components of the CAPS software base.

4. The conceptual methodologies of CAMP and CAPS are very similar. Comparison of the two approaches is useful for showing the areas of commonality and difference between a prototyping environment and an environment oriented toward software generation. Additionally, comparison of the approaches to composition used in CAPS and CAMP may be useful in development of the rules and methods used in the CAPS Software Base for interacting with the object-oriented database of reusable components.

This section presents a summary of the CAMP approach and brief descriptions of the highlights of research conducted as part of this project that merit further study for

possible incorporation into the CAPS environment. Included in this discussion are the methods used by the developers to define the application domain of the system, the methods considered and finally chosen for the construction of the reusable software components in CAMP, the cataloging of components for storage in a software library, and the proposed design of an expert system which will be used to perform the composition and translation of specifications into programs. This final system is still being developed; but, expert assistance is currently in use for the catalog component of the library to assist in search and retrieval of components.

## A. DOMAIN ANALYSIS IN CAMP

In Chapter II the area of domain analysis was defined as an approach that can be used to define domain related categories of operations, objects, and structures [Ref. 1: p. 15]. Results of the domain analysis can be incorporated into classification schemes, domain languages used by expert systems, and even the design of individual components used to make up the classes of reusable components. The taxonomy of reusable parts for the CAMP project was presented as an example of one of these products. Domain analysis was reflected in all areas of the project. There were three steps involved in the domain analysis performed for this project:

1. Domain Definition: The process of determining the scope of the domain analysis. This includes an analysis and formal definition of the boundaries of the domain, an analysis of overlapping areas which may be included in the domain but are also appropriate in other areas, and areas of intersection between application or domain independent areas and domain specific areas.

2. Domain Representation: The selection of a set of applications used to characterize the domain under investigation.

3. Commonality Study: Analysis of previous domain related implementations to identify the common objects, operations and structures that are candidates for construction as reusable software components.

Domain definition requires direction from the projected users. It is important that the domain definition and boundaries be established early in the domain analysis process to avoid the problems of expanding beyond the boundaries into overlapping domains. Once the domain is defined a set of common applications within the domain of interest are selected for analysis. Again this activity involves the users of projected systems as well as previously developed systems. The quality of the remaining domain analysis and the time involved in completing the analysis can be greatly influenced by the quality and availability of system documentation and source code. The CAMP developers found that requirements and design information were much more valuable than the source code in the process. These documents were also much less likely to be up to date or even available at all for older systems.

The final stage of the domain analysis, the Commonality study looked at the commonality of the selected missile systems. A functional strip method was used to analyze particular functions. This method involves the examination of the actual implementations of common functions performed in the majority of the systems looking

for commonality of both the specific implementations and the supporting parts of the subsystem being examined [Ref. 1: pp. 21-23].

One of the more interesting concepts exploited in the CAMP project was the identification of vertical and horizontal domains within the application area [Ref 1: pp. 17-18]. Vertical domains are application dependent groupings of software systems most closely related to subsystems that they represent or support. Horizontal domains are application independent groupings with the common objects and functions representing such categories as abstract data structures, mathematical functions and other relatively common categories. These two domain types allow the identification of components in multiple functional areas based on the intersections between the two types of domains. The taxonomy presented earlier represents groupings of components based on both types of domains. The following is a summary of the groupings by domain type:

Horizontal Domains: Data Package Parts, Abstract Mechanism Parts, Process Management Parts, Mathematical Parts, General Utility Parts.

Vertical Domains: Equipment Interface Parts, Primary Operation Parts.

The result of the domain analysis conducted as part of the CAMP project was the development of parts in the horizontal domains that supported the subsystem level objects and functions required by the vertical domains.

## B. ADA LANGUAGE CONSIDERATIONS

Ada is the target implementation language for CAMP parts. The major Ada constructs considered in the implementations were the package and Ada generic capabilities. The goal for implementation of the parts was to use packages to group related groupings of code and, where possible, make the parts generic. In their analysis of the design methods to be used to implement the reusable parts they considered six methods which are summarized below [Ref. 1: pp. 38-50]

1. Typeless method: In this method all data objects and actual parameters are of type float. Alleviates the need for special mathematical operators and functions since they are all defined in standard packages for type float. Its severe disadvantage is that type checking cannot be done by the compiler and runtime system because all objects are of the same type.

2. Overloaded Method: Provides a user with a single package containing multiple implementations of the specification using the various allowable data types. The Ada language allows this overloading and it would be extremely simple. However, it would require the implementation/reimplementation of the same function many times to accommodate all combinations of desired data types.

3. Generic Method: Use of the Ada generic facility to define a part that can be instantiated using user defined types. Again, this method takes advantage of an important feature of the Ada language, but places the burden of supplying the instantiation information on the user. Its main advantage is flexibility in the use of data types.

4. State Machine Method: Defining "black box" objects having a single set of external interfaces with operations which permit a user of the machine to examine the internal state of the machine or change the state of the machine. Alleviates the data typing and mathematical problems of the other methods. But adds the need to convert all data to the part's internal format which can result in additional overhead.

5. Abstract Data Type Method: Similar to the State machine method but less flexible in implementation. The types must be included in both the specification and the body portions of the Ada parts. Would require more implementations to provide similar capabilities to those of the State Machine.

6. Skeletal Code Method: Development of templates which may be manipulated using manual methods or an expert system to build components from user supplied types and required functions as well as other parts. This method was discarded primarily due to the complexity involved in implementation of this approach using current technology. As expert systems improve this approach may become more viable.

The selected approach for design of CAMP parts were the generic and overloaded methods described above. There are however, specific instances of abstract data types state machines, and schematic included in the CAMP parts that were ultimately developed.

## C. THE PARTS CATALOG

As part of the project a great deal of effort was placed on the identification of attributes and documentation on individual parts that can be stored in a cataloging system. [Refs. 22-25] The conclusions of the research in this area were that the catalog developed during the building of a software parts library should contain the necessary attributes needed to find the components as well as documentation needed to produce required documents as part of the software development process. They developed a sample listing of attributes and implemented a catalog system using a relational database as part of the project. The discussion of this area contained in the overview document contains

information and references that are very important in the incorporation of documentation capabilities into the CAPS system.

## D. THE ADA MISSILE PARTS ENGINEERING EXPERT SYSTEM (AMPEE)

During this research an expert system was designed and prototypes are continually being improved that will result in a composition based software generation system for the missile systems domain. An extensive amount of information on the goals of expert systems in software generation is presented as well as an overview of many of the more prominent efforts in this area. The approach used for integration of expert systems into the component retrieval from the parts catalog described in this thesis are conceptually very similar. The CAMP developers also outline many other issues not included in this thesis. Their system is designed to support the composition of parts in a different way than that used by CAPS. Their research does seem to confirm that the method of composition or generation of components is the major determining factor in determining the form of the software base and expert systems support used to incorporate reusable components into a program.

## E. OVERALL UTILITY OF THE CAMP PROJECT

The CAMP Project provides a source of detailed research on many of the technical and managerial issues related to development of software generation systems in general and software generation systems using reusable components and composition based strategies in particular. The documentation provided by the project is a valuable source

of overview information and provides many pointers to detailed information on the subject area. There have been over 400 actual parts developed to date. The descriptive information provided with these parts [Refs. 22-25] was analyzed as part of this thesis. The parts provide many functions which may be useful in building a sample domain for use in CAPS.

Applying the classification scheme described in this thesis to the parts described in CAMP revealed that at the subsystem level, or within the vertical domains, a relatively even distribution of components is possible. However for the horizontal domains there was a tendency for the parts to be clustered in the generic function area. It may be possible to more evenly distribute these components in a CAPS reusable software base by providing instantiation information in the specific classes for the various components in the horizontal domains. This would add to the number of parts available in the CAPS software base, with a minimum of actual source code being required for these new specific objects (instantiation code only). The generic objects would be referenced in these new objects and would remain intact.

The CAMP project provides a source of information and software that has the potential to greatly expand the current software base used by CAPS. Detailed examination of the parts is still required. But, the initial analysis shows that these parts may form the basis of the expanded software base to support CAPS.

# APPENDIX B. THE PSDL GRAMMAR

This grammar uses standard symbology conventions. {Curly Braces} enclose items which may appear zero or more times. [Square Brackets] enclose items which may appear zero or one time in a rule. **Bold Face** items are terminal keywords. Items contained in "Double Quotes" are character literals. The "|" vertical bar indicates a list of options from which no more than one item may be selected. This grammar represents the updated version of the PSDL grammar as of 20 June 1990.

Start = psdl

psdl = {component}

component = data_type | operator

data_type = **type** id type_spec type_impl

operator = **operator** id operator_spec operator_impl

type_spec = **specification** [generic_param] [type_decl]

{**operator** id operator_spec} [functionality] **end**

type_impl = **implementation ada** id "{" text "}" **end**

| **implementation** type_name

{**operator** id operator_impl} **end**

operator_spec = specification {interfa . } [functionality] **end**

operator_impl = **implementation ada** id "{" text "}"

| **implementation** psdl_impl

type_decl =
id_list ":" type_name {"," id_list ":" type_name}

functionality = [keywords] [informal_desc] [formal_desc]

psdl_impl = data_flow_diagram [streams] [timers] [control_constraints] [informal_desc]

**end**

type_name = id |

id "[" actual_parameter_list "]" |

id "[" type_decl "]"

actual_parameter_list = actual_parameter

{ "," actual_parameter }

.tual_parameter = type_name | expression

interface = attribute [reqmts_trace]

id_list = id {"," id}

keywords = **keywords** id_list

informal_desc = **description** "{" text "}"

formal_desc = **axioms** "{" text "}"

data_flow_diagram = **graph** {vertex} {edge}

streams = **data stream** type_decl

timers = **timer** id_list

attribute = input
| output
| generic_param
| states
| exceptions
| timing_info

input = **input** type_decl

output = **output** type_decl

generic_param = **generic** type_decl

states = **states** type_decl **initially** expression_list

exceptions = **exceptions** id_list

timing_info = [**maximum execution time**]
[**minimum calling period** time]
[**maximum response time** time]

reqmts_trace = **by requirements** id_list

vertex = **vertex** op_id [":"time]

edge = **edge** id [":"time] op_id "->" op_id

op_id = id ["(" [id_list] "|" [id_list] ")"]

control_constraints = **control constraints** {constraint}

constraint = **operator** id
[**triggered** (trigger | [trigger] **if** predicate) [reqmts_trace]]
[**period** time [reqmts_trace]]
[**finish within** time [reqmts_trace]]
{constraint_options}

```
trigger = by all id_list
        | by some id_list

constraint_options =
          output id_list if predicate [reqmts_trace]
        | exception id [if predicate] [reqmts_trace]
        | timer_op id [if predicate] [reqmts_trace]

timer_op = read timer
        | reset timer
        | start timer
        | stop timer

expression_list = expression {"," expression}

time = integer [unit]

unit = ms | sec | min | hours


expression = constant
        | id
        | type_name "." id "(" expression_list ")"

predicate = simple_expression
        | simple_expression rel_op simple_expression


simple_expression = [sign] integer [unit]
        | [sign] real
        | [not] id
        | string
        | [not] "(" predicate ")"
        | [not] boolean_constant

bool_op = and | or
```

rel_op = "<" | "<=" | ">" | ">=" | "=" | "/=" | ":"

real = integer "." integer

integer = digit{digit}

boolean_constant = **true** | **false**

numeric_constant = real | integer

constant = numeric_constant | boolean_constant

sign = "+" | "-"

char = any printable character except "}"

digit = "0 .. 9"

letter = "a .. z" | "A .. Z" | "_"

alpha_numeric = letter | digit

id = letter{alpha_numeric}

string = """ text """

text = {char}

## LIST OF REFERENCES


1. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Volume I.: Overview and Commonality Study Results*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, May, 1986.

2. Booch, G., *Software Components with Ada*, The Benjamin/Cummings Publishing Company Inc., Menlo Park, California, 1987.

3. Barr, A., Cohen, P. R. and Feigenbaum, E. A. eds, *The Handbook of Artificial Intelligence Volume IV*, Addison Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

4. Biggerstaff, T. J. and Perlis, A.J., eds. *Software Reusability, Volume I, Concepts and Models*, ACM Press, New York, New York, 1989.

5. White, L.J., *The Development of a Rapid Prototyping Environment*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1989.

6. Galik, D.A., *A Conceptual Design of A Software Base Management System for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December, 1988.

7. Steigerwald, R. A., *Reusable Software Components*, M.S. Thesis, University of Illinois at Urbana-Champaign, 1986.

8. Biggerstaff, T. J. and Perlis, A.J., eds. *Software Reusability, Volume II, Applications and Experience*, ACM Press, New York, New York, 1989.

9. Coello, E. M. P., *Cognitive Issues in Software Reuse*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June, 1985.

10. Luqi, and Ketabchi, M., *A Computer Aided Prototyping System*, IEEE Software, March, 1988, pp. 66-72.

11. Luqi, *Software Evolution Through Rapid Prototyping*, IEEE Computer, May 1989, pp. 13-25.

12. Luqi, Berzins, V., and Yeh, R., *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, October, 1988, pp. 1409-1423.

13. Gupta, R., Cheng, W., Gupta, R., Hardonag, I., and Breuer, M.A., *An Object-Oriented VLSI CAD Framework*, IEEE Computer, May 1989, pp. 28-37.

14. Kim, W., and Lochovsky, F. H., eds., *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, New York, 1989.

15. Comer, E.R., *Ada Box Structures Methodology Handbook*, Software Productivity Solutions Inc., Melbourne, Florida, July, 1989.

16. Berzins, V., and Luqi, *Semantics of a Real Time Language*, Technical Report, Naval Postgraduate School, Monterey, California, September, 1988.

17. Department of Defense, *Reference Manual for the Ada Programming Language*, ANSI/MIL-STD-1815A, U.S. Department of Defense, 1983.

18. Ontologic Incorporated., *Ontos Database System Documentation*, Ontologic Inc., Burlington, Massachusetts, March, 1990.

19. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Composition System Vol 1.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

20. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Composition System Vol 2.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

21. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Composition System Vol 3.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

22. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 1.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

23. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 2.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

24. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 3.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

25. McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 4.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

26. Conn, R., *The Ada Software Repository and the Defense Data Network*, Zoetrope Publishing Co., Inc., New York, New York, 1987.

## BIBLIOGRAPHY


Altizer, C., *Implementation of a Language Translator for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December, 1988.

Association for Computing Machinery, *OOPSLA '86 Conference Proceedings*, Special Issue of SIGPLAN Notices, Vol. 21, No. 11, November, 1986.

Association for Computing Machinery, *OOPSLA '87 Conference Proceedings*, Special Issue of SIGPLAN Notices, Vol. 22, No. 12, December, 1987.

Association for Computing Machinery, *OOPSLA '87 Addendum to the Proceedings*, Special Issue of SIGPLAN Notices, Vol. 23, No. 5, May, 1988.

Barr, A. and Feigenbaum, E. A., Editors, *The Handbook of Artificial Intelligence, Volume 1*, HeurisTech Press, Stanford, California, 1981.

Barr, Avron, Cohen, Paul R. and Feigenbaum, Edward A. eds., *The Handbook of Artificial Intelligence Volume IV*, Addison Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

Biggerstaff, T. J. and Perlis, A. J., eds. *Software Reusability, Volume I, Concepts and Models*, ACM Press, New York, New York, 1989.

Biggerstaff, T. J. and Perlis A. J., eds. *Software Reusability, Volume II, Applications and Experience*, ACM Press, New York, New York, 1989.

Booch, Grady. *Software Components with Ada*, The Benjamin/Cummings Publishing Company Inc., Menlo Park, California, 1987.

Booch, Grady. *Software Engineering with Ada*, The Benjamin/Cummings Publishing Company Inc., Menlo Park, California, 1983.

Booch, G., *Object Oriented Development*, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February, 1986.

Brackett, J.W., *Software Requirements*, SEI Curriculum Module SEI-CM-19-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, December 1988.

Cashman M., *Object-Oriented Domain Analysis*, Software Engineering Notes, Vol. 14, No. 6, October, 1989.

Coello, E. M. P., *Cognitive Issues in Software Reuse*, M.S. Thesis, Naval Postgraduate School, Monterey, California, June, 1985.

Comer, E.R., *Ada Box Structures Methodology Handbook*, Software Productivity Solutions Inc., Melbourne, Florida, 31 July, 1989.

Conn, R., *The Ada Software Repository and the Defense Data Network*, Zoetrope Publishing Co., Inc., New York, New York, 1987.

Davis, A.M., *A Comparison of Techniques for the Specification of External System Behavior*, Communications of the ACM, vol. 31, no. 9, September, 1988.

Department of Defense, *Military Standard for Defense System Software Development*, DOD-STD-2167A, U.S. Department of Defense, 1987.

Douglas,B.S., *A Conceptual Level Design of a Design Database for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December, 1988.

Galik, D.A., *A Conceptual Design of A Software Base Management System for the Computer Aided Prototyping System*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1988.

Gupta, R., Cheng, W., Gupta, R., Hardonag, I., and Breuer, M.A., *An Object-Oriented VLSI CAD Framework*, IEEE Computer, May, 1989.

Jordan, P.W., Keller, K.S., Tucker, R.W., and Vogel, D., *Software Storming*, IEEE Computer, May, 1989.

Kim, W., and Lochovsky, F. H., eds., *Object-Oriented Concepts, Databases, and Applications*, ACM Press, New York, New York, 1989.

Luqi, *Software Evolution Through Rapid Prototyping*, IEEE Computer, May, 1989.

Luqi, and Ketabchi, M., *A Computer Aided Prototyping System*, IEEE Software, March, 1988.

Luqi, Berzins, V., and Yeh, R., *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, October, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Volume I.: Overview and Commonality Study Results*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, May, 1986.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 1.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 2.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 3.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Catalog. Vol 4.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Composition System Vol 1.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Composition System Vol 2.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. Parts Composition System Vol 3.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

McDonnell Douglas Astronautics Co., *Common Ada Missile Packages (CAMP) Project. 11th Missile Application. Volume 2. Top-Level Design.*, McDonnell Douglas Astronautics Co., St. Louis, Missouri, March, 1988.

Meyer, B., *Reusability: The Case for Object-Oriented Design*, IEEE Software, Vol. 4, No. 2, March, 1987.

Mills, H.D., Linger, R.C., and Hevner, A.R., *Box Structured Information Systems*, IBM Systems Journal, vol 26., no. 4, 1987.

Parnas D.L., *On the Criteria To Be Used in Decomposing Systems Into Modules*, Communications of the ACM, Vol. 5,No. 12, December, 1972.

Seidewitz, E.. *General Object-Oriented Software Development: Background and Experience*, Journal of Systems and Software vol. 9, 1989.

Shlaer, S. and Mellor, S.J., *An Object-Oriented Approach to Domain Analysis*, Software Engineering Notes, Vol. 14, No. 5, July, 1989.

Steigerwald, R. A., *Reusable Software Components,* M.S. Thesis, University of Illinois at Urbana-Champaign, 1986.

Tsai, J. and Ridge, J.C., *Intelligent Support for Specifications Transformation*, IEEE Software, November, 1988.

White, L. J., *The Development of a Rapid Prototyping Environment*, M.S. Thesis, Naval Postgraduate School, Monterey, California, December, 1989.

## Initial Distribution List

| | |
|---|---|
| Defense Technical Information Center<br>Cameron Station<br>Alexandria, VA 22304-6145 | 2 |
| Dudley Knox Library<br>Code 0142<br>Naval Postgraduate School<br>Monterey, CA 93943 | 2 |
| Director of Research Administration<br>Code 012<br>Naval Postgraduate School<br>Monterey, CA 93943 | 1 |
| Chairman, Code 52<br>Computer Science Department<br>Naval Postgraduate School<br>Monterey, CA 93943 | 1 |
| Office of Naval Research<br>800 N. Quincy Street<br>Arlington, VA 22217-5000 | 1 |
| Center for Naval Analysis<br>4401 Ford Avenue<br>Alexandria, VA 22302-0268 | 1 |
| National Science Foundation<br>Division of Computer and Computation Research<br>Washington, D.C. 20550 | 1 |
| Office of the Chief of Naval Operations<br>Code OP-941<br>Washington, D.C. 20350 | 1 |
| Office of the Chief of Naval Operations<br>Code OP-945<br>Washington, D.C. 20350 | 1 |

Commander Naval Telecommunications Command 2
Naval Telecommunications Command Headquarters
4401 Massachusetts Avenue NW
Washington, D.C. 20390-5290

Commander Naval Data Automation Command 1
Washington Navy Yard
Washington, D.C. 20374-1662

Dr. Lui Sha 1
Carnegie Mellon University
Software Engineering Institute
Department of Computer Science
Pittsburgh, PA 15260

COL C. Cox, USAF 1
JCS (J-8)
Nuclear Force Analysis Division
Pentagon
Washington, D.C. 20318-8000

Commanding Officer 1
Naval Research Laboratory
Code 5150
Washington, D.C. 20375-5000

Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ISTO)
1400 Wilson Boulevard
Arlington, VA 22209-2308

Defense Advanced Research Projects Agency (DARPA) 1
Director, Naval Technology Office
1400 Wilson Boulevard
Arlington, VA 2209-2308

Defense Advanced Research Projects Agency (DARPA) 1
Director, Prototype Projects Office
1400 Wilson Boulevard
Arlington, VA 2209-2308

Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, VA 2209-2308

Dr. R. M. Carroll (OP-01B2) 1
Chief of Naval Operations
Washington, DC 20350

Dr. Aimram Yehudai 1
Tel Aviv University
School of Mathematical Sciences
Department of Computer Science
Tel Aviv, Israel 69978

Dr. Bernd Kraemer 1
GMD Postfach 1240
Schloss Birlinghaven
D-5205
Sankt Augustin 1, West Germany

Dr. Robert M. Balzer 1
USC-Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina del Ray, CA 90292-6695

Dr. Ted Lewis 1
Editor-in-Chief, IEEE Software
Oregon State University
Computer Science Department
Corvallis, OR 97331

IBM T.J.Watson Research Center 1
Attn. Dr. A. Stoyenko
P.O. Box 704
Yorktown Heights, NY 10598

Dr. R. T. Yeh
International Software Systems Inc. 1
12710 Research Boulevard, Suite 301
Austin, TX 78759

Attn. Dr. C. Green 1
Kestrel Institute
1801 Page Mill Road
Palo Alto, CA 94304

Prof. D. Berry 1
Department of Computer Science
University of California
Los Angelas, CA 90024

Dr. B. Liskov 1
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
545 Tech Square
Cambridge, MA 02139

Dr. J. Guttag 1
Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science
545 Tech Square
Cambridge, MA 02139

Director, Naval Telecommunications System Integration Center 1
NAVCOMMUNIT Washington
Washington, D.C. 20363-5110

Space and Naval Warfare Systems Command 1
Attn: Dr. Knudsen, Code PD50
Washington, D.C. 20363-5110

Ada Joint Program Office 1
OUSDRE(R&AT)
The Pentagon
Washington, D.C. 23030

CAPT A. Thompson 1
Naval Sea Systems Command
National Center #2, Suite 7N06
Washington, D.C. 22202

Dr. Peter Ng 1
New Jersey Institute of Technology
Computer Science Department
Newark, NJ 07102

Dr. Van Tilborg 1
Office of Naval Research
Computer Science Division, Code 1133
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. R. Wachter 1
Office of Naval Research
Computer Science Division, Code 1133
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. J. Smith, Code 1211 1
Office of Naval Research1
Applied Mathematics and Computer Science
800 N. Quincy Street
Arlington, VA 22217-5000

Dr. R. Kieburtz 1
Oregon Graduate Center1
Portland (Beaverton)
Portland, OR 97005

Dr. M. Ketabchi 1
Santa Clara University
Department of Electrical Engineering and Computer Science
Santa Clara, CA 95053

Attn. Dr. L. Belady 1
Software Group, MCC
9430 Research Boulevard
Austin, TX 78759

Attn. Dr. Murat Tanik 1
Southern Methodist University
Computer Science and Engineering Department
Dallas, TX 75275

Dr. Ming Liu 1
The Ohio State University
Department of Computer and Information Science
2036 Neil Ave Mall
Columbus, OH 43210-1277

Mr. William E. Rzepka 1
U.S. Air Force Systems Command
Rome Air Development Center
RADC/COE
Griffis Air Force Base, NY 13441-5700

Dr. C.V. Ramamoorthy 1
University of California at Berkeley
Department of Electrical Engineering and Computer Science
Computer Science Division
Berkeley, CA 90624

Dr. Nancy Levenson 1
University of California at Irvine
Department of Computer and Information Science
Irvine, CA 92717

Dr. Mike Reiley 1
Fleet Combat Directional Systems Support Activity
San Diego, CA 92147-5081

Dr. William Howden 1
University of California at San Diego
Department of Computer Science
La Jolla, CA 92093

Dr. Earl Chavis (OP-162) 1
Chief of Naval Operations
Washington, DC 20350

Dr. Jane W. S. Liu 1
University of Illinois
Department of Computer Science
Urbana Champaign, IL 61801

Dr. Alan Hevner
University of Maryland
College of Business Management
Tydings Hall, Room 0137
College Park, MD 20742 1

Dr. Y. H. Chu
University of Maryland1
Computer Science Department
College Park, MD 20742 1

Dr. N. Roussapoulos
University of Maryland
Computer Science Department
College Park, MD 20742 1

Dr. Alfs Berztiss
University of Pittsburgh
Department of Computer Science
Pittsburgh, PA 15260 1

Dr. Al Mok
University of Texas at Austin
Computer Science Department
Austin, TX 78712 1

George Sumiall
US Army Headquarters
CECOM
AMSEL-RD-SE-AST-SE
Fort Monmouth, NJ 07703-5000 1

Attn: Joel Trimble
1211 South Fern Street, C107
Arlington, VA 22202 1

Naval Ocean Systems Center
Attn: Linwood Sutton, Code 423
San Diego, CA 92152-5000 1